TIBCO Business Studio™ Customization

Version 4.3.2

May 2022

Contents

Understanding the Process Package File Format

The TIBCO Business Studio package file uses the following schemas:

- XPDL Version 2.1
- XPDL Version 1.0 (Internally, for direct deployment to the TIBCO iProcess Engine)
- Extensions schema - process-related XPDL extensions
- Simulation schema - XPDL extensions to support the simulation of Processes

To import a file, only the correct XPDL and Extensions semantics are required; simulation is optional and default simulation parameters can be added by TIBCO Business Studio once the process has been imported.

- Various extension schemas, each describing meta-data for one or more sub-types of service task.

This section describes the extensions and simulation schemas used by TIBCO Business Studio.

Implementing Business Process Integration (Import)

Scenario: TIBCO Business Studio is not yet the primary corporate modeling tool.

Goal: To leverage the investment in the current Business Process Analysis (BPA) tool by importing existing processes into TIBCO Business Studio and getting the processes into an executable form.

To import a process from an established BPA tool into TIBCO Business Studio, you must do the following:

Procedure

1. Study the selected BPA tool, particularly the XML export facility.
2. Identify the significant elements in the BPA tool's XML format.
3. Referring to Understanding the Process Package File Format, familiarize yourself with the XPDL format and the TIBCO extensions.
4. Construct a mapping table (for example, a spreadsheet) between the interesting elements. This will be the basis of the XSLT that will transform elements from the BPA XML output into XPDL that can be imported into TIBCO Business Studio.
5. Implement the mapping in XSLT.
6. Register the XSLT as a plug-in that may be distributed to all users (by copying to the `plugins` directory) or installed directly. See Creating an XSLT-Based Import Export Wizard.

Exporting from TIBCO Business Studio

Scenario: Application developer or systems integrator wants to take output from TIBCO Business Studio and embed it within a custom BPM solution (for example, create a JSP web page for each user task).

Goal: Transform part or all of the process into supporting artifacts in the overall solution (for example, JSP forms, portal pages and document management systems).

To export process information from TIBCO Business Studio to another application:

Procedure

1. Referring to Understanding the Process Package File Format, familiarize yourself with the XPDL format and the TIBCO extensions.
2. Study the required output.

For example, in the case of JSP forms, suppose that for each user task in the Process, the desired output is a default form with a field for each parameter of the user task.

3. Construct a mapping table (for example, a spreadsheet) between the significant TIBCO Business Studio XPDL elements and their target objects (be they XML objects or widgets in a GUI toolkit).
4. Implement the mapping in XSLT.
5. (Optional) Document any restrictions on creating the output (such as the JSP form generator supporting only simple data field types).
6. Register the XSLT as a plug-in that may be distributed to all users (by copying to the `plugins` directory) or installed directly.

   See Creating an XSLT-Based Import Export Wizard.

XPDL 2.1 Schema Overview

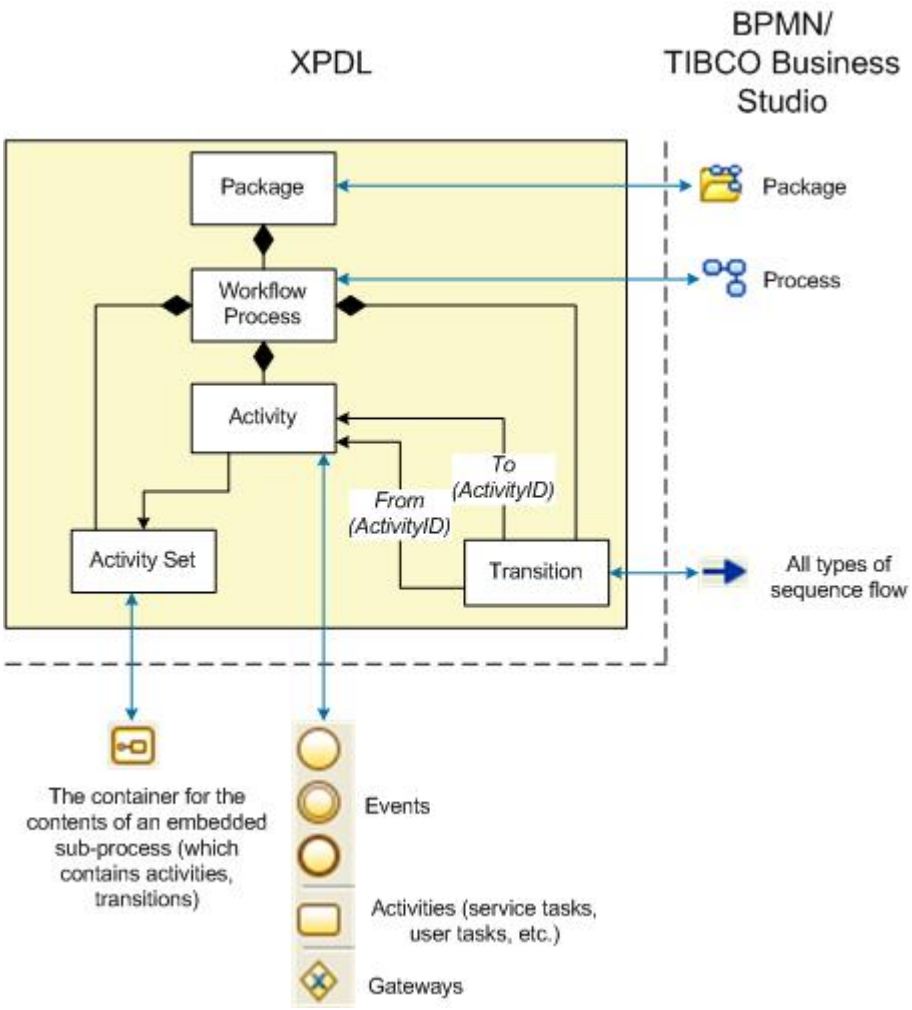
The XPDL Version 2.1 schema provides all the elements needed to execute a process as well as the visual elements necessary to view a process in TIBCO Business Studio. The following series of diagrams show the relationship between the BPMN elements in TIBCO Business Studio and their mapping to the XPDL schema.

For more information, see:

- XPDL - http://www.wfmc.org/standards/xpdl.htm
- BPMN - http://www.bpmn.org/

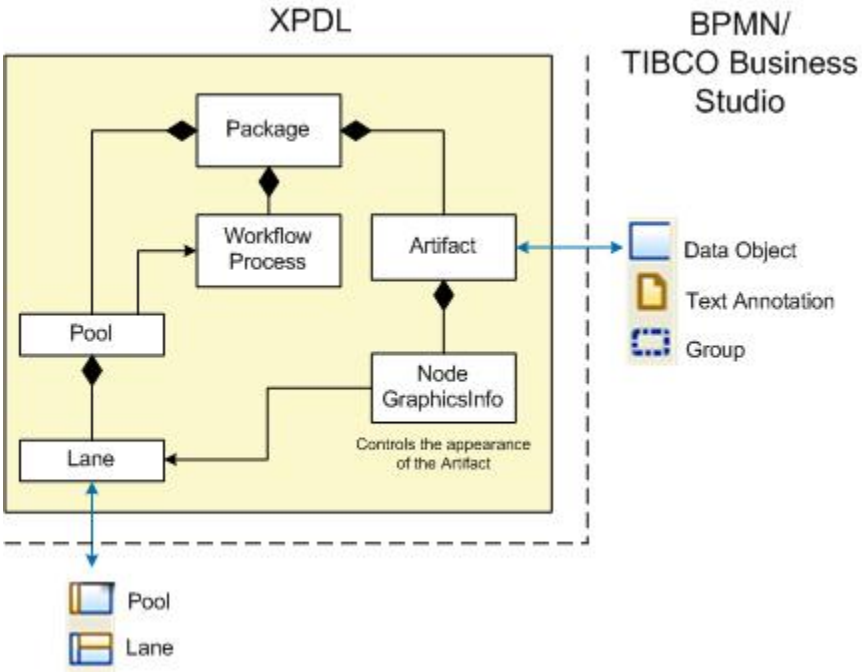
Process Flow Objects

This diagram shows the major process flow objects (package, process, activity and transitions), excluding the visual objects such as pools, lanes:



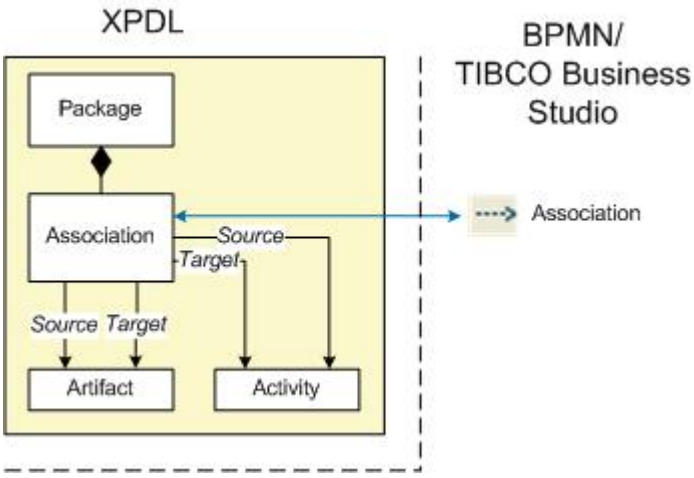
Artifacts

XPDL artifacts belong to the package and are referred to in a process by lanes and pools. this diagram shows the relationship of artifacts to other objects:



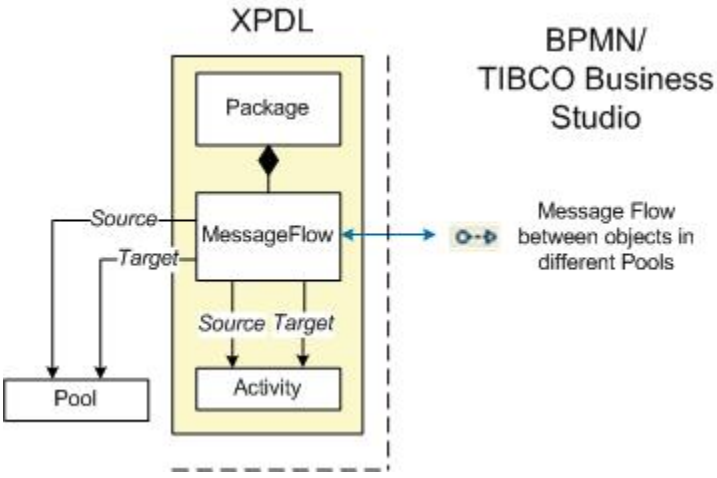
Associations

This diagram shows how optional associations are made between non-flow objects (such as a text annotation) and flow objects or flows:



Message Flow

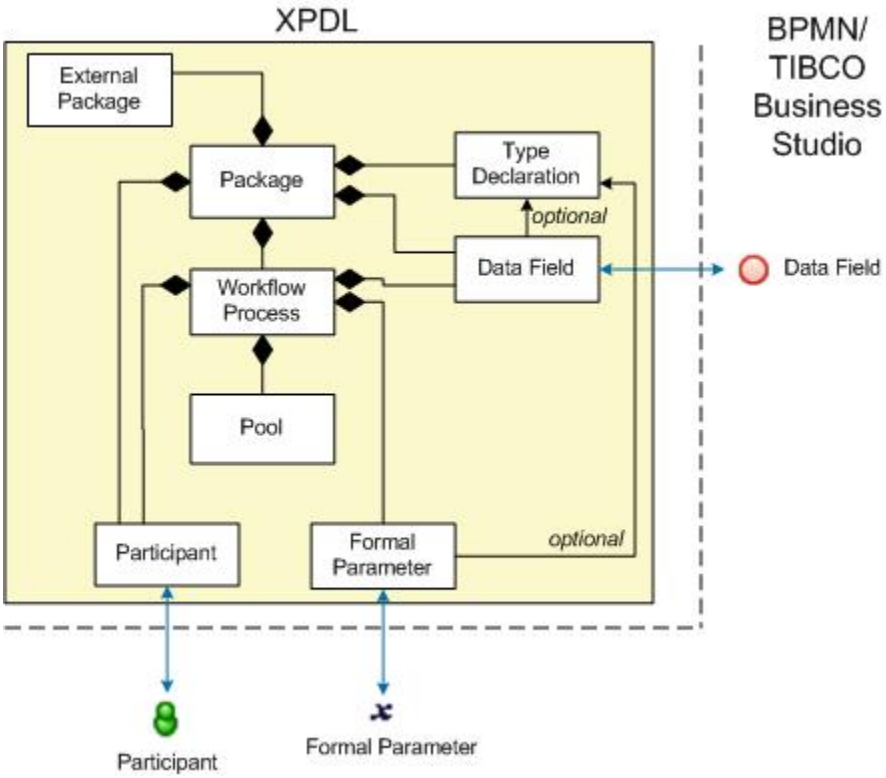
Message flow can connect objects in different pools or objects in one pool with the boundary of another pool. the following diagram shows message flow:



Data Fields, Formal Parameters, Participants, and Type Declarations

Data fields and participants can be scoped in that they can be created at the process level or at the package level (if you want to share them amongst processes). Formal parameters can only be created at the process level and type declarations can only be created at the package level.

The following diagram shows the relationship of data fields, formal parameters, participants, and type declarations to pools, process, and package:



The relationship of a participant to a pool is denoted in BPMN, but neither required nor enforced by TIBCO Business Studio. A BPMN participant is always represented by a pool. However, an XPDL participant describes a resource that can perform an activity and is not necessarily represented by a pool.

XPDL Migration Injector

The XPDL Migration Injector is implemented in the `com.tibco.xpd.analyst.resources.xpdl2.xpdlMigrationInjector` plugin.

The XPDL Migration Injector allows a contributor to inject an extra xslt transformation into the sequence of migration xslts used to bring a TIBCO Business Studio XPDL process package file from its defined format version to the current format version of TIBCO Business Studio.

Each TIBCO Business Studio XPDL contains the format version extended attribute of the version of TIBCO Business Studio it was created with or for. If the format version is less than that in current use by the version of TIBCO Business Studio in question it is migrated to the latest format version via a sequence of xslt transformations. Each transformation increases the format version by 1 and performs the necessary transformations to convert the file from one format version to the next format version.

The contributor can decide upon a format version (which governs at what point in the migration sequence it is executed) and whether it should be executed before or after the built in TIBCO Business Studio transformation between the 2 versions.

Great care should be taken to ensure that undesirable side effects are avoided and that unaffected XPDL elements are output from the transformation unchanged.

Configuration Markup

```
<!ELEMENT extension (migrationInjector)+>
<!ATTLIST extension
point CDATA #REQUIRED
id    CDATA #IMPLIED
name  CDATA #IMPLIED>
<!ELEMENT migrationInjector EMPTY>
<!ATTLIST migrationInjector
beforeOrAfter (Before|After) "Before"
formatVersion CDATA "1"
xsltFile      CDATA #REQUIRED>
```

Each migration injector can inject one xslt transformation before or after migration to a given TIBCO Business Studio XPDL format version.

`beforeOrAfter` - Whether the xslt should be injected and executed before or after the standard TIBCO Business Studio migration to the given formatVersion.

`formatVersion` - Format version - effectively the position in XPDL migration sequence where to inject the xslt.

`xsltFile` - Xslt transformation to perform just before or after standard migration conversion to a given `formatVersion` from the previous `formatVersion`.

Execution Details of a Package

This section describes the parts of the XPDL package file that must be present if you want to execute a TIBCO Business Studio XPDL package/process in BPM or iProcess Engine.

It does not cover visual aspects of a process (see Visual Characteristics of a Package).

Creating a Package

The root XPDL element is **Package** and the following namespaces are used:

```
<?xml version="1.0" encoding="UTF-8"?>
<xpdl2:Package xmlns:xpdExt="http://www.tibco.com/XPD/xpdExtension1.0.0"
xmlns:xpdl2="http://www.wfmc.org/2008/XPDL2.1" Id="_OU85UG0fEd29y-F9Fx-OGw"
xpdExt:DisplayName="ProcessPackage" Name="ProcessPackage">
```

The semantics for the **Package**, **Process** and **Redefineable Headers** allow the least restrictive interpretation of the XPDL schema possible.

A **Package Header** in TIBCO Business Studio is defined by XPDL as follows:

```
<xpdl2:PackageHeader xpdExt:Language="en_GB">
  <xpdl2:XPDLVersion>2.1</xpdl2:XPDLVersion>
  <xpdl2:Vendor>TIBCO</xpdl2:Vendor>
  <xpdl2:Created>2008-08-18T14:14:47BST</xpdl2:Created>
  <xpdl2:Description></xpdl2:Description>
  <xpdl2:Documentation></xpdl2:Documentation>
  <xpdl2:CostUnit>GBP</xpdl2:CostUnit>
</xpdl2:PackageHeader>
```

TIBCO Business Studio contributes the following elements to the Package only:

Attribute	Example	Notes
CreatedBy	`<xpdl2:ExtendedAttribute Name="CreatedBy" Value="TIBCO Business Studio"/>`	You can provide any value. The example indicates that the process was created in TIBCO Business Studio, but there is no checking or validation.
FormatVersion	`<xpdl2:ExtendedAttribute Name="FormatVersion" Value="8"/>`	*Mandatory*. The FormatVersion must be set to match the version of TIBCO Business Studio you are working with. When importing into newer versions, TIBCO Business Studio has a Quick Fix that allows you to migrate to the latest FormatVersion, thus providing a level of backwards compatibility in new versions of TIBCO Business Studio without the necessity to change your import.

These are also elements called **ExtendedAttribute** that are in the base namespace. For example:

```
<xpdl2:Task>
<xpdl2:TaskService xpdExt:ImplementationType="E-Mail"
    xpdl2:Implementation="Other">
      <xpdl2:MessageIn Id="_V_wdICrAEdy4JvZXy-dYnw"/>
         <xpdl2:MessageOut Id="_V_wdISrAEdy4JvZXy-dYnw"/>
         <email:Email>
            <email:Definition>
              <email:From email:Configuration="Server"/>
              <email:To>manager@foo.com</email:To>
              <email:Subject>RE: minutes of our
                    meeting</email:Subject>
            </email:Definition>
            <email:SMTP email:Configuration="Server"/>
         </email:Email>
</xpdl2:TaskService>
</xpdl2:Task>
```

Setting the Destination Environment

Setting the destination environment on a process controls the validation that is performed on that process and also, in the case of simulation, what simulation parameters are associated with the process. You can specify multiple destination environments.

The following example shows the XPDL for a process intended for simulation:

```
<xpdl2:ExtendedAttribute Name="Destination"        Value="Simulation"/>
```

TIBCO Business Studio provides several destination environments, including BPM and Simulation.

Value	Description
BPM	Specifies that the process is validated for direct deployment to the TIBCO BPM 1.0.
Simulation	Specifies that the process is validated for simulation.
iProcess	Specifies that the process is validated for direct deployment to the iProcess Engine 10.5 or higher.

The specific "destination components" that make up these destination environments can be customized in the Preferences. For more information about destination components, see *TIBCO Business Studio iProcess Implementation Guide* or *TIBCO Business Studio BPM Implementation Guide*.

Date and Time Data Types

TIBCO Business Studio now supports separate date and time data fields, as well as the previous combined datetime data field:

```
<xpdl2:DataFields>
  <xpdl2:DataField Id="_PEzc0HQNEd2Pfrjp77cFsg"
  xpdExt:DisplayName="Date" Name="Date">
     <xpdl2:DataType>
       <xpdl2:BasicType Type="DATE"/>
     </xpdl2:DataType>
    <xpdl2:InitialValue></xpdl2:InitialValue>
  </xpdl2:DataField>
   <xpdl2:DataField Id="_RvJjMHQNEd2Pfrjp77cFsg"
   xpdExt:DisplayName="Time" Name="Time">
     <xpdl2:DataType>
       <xpdl2:BasicType Type="TIME"/>
     </xpdl2:DataType>
     <xpdl2:InitialValue></xpdl2:InitialValue>
  </xpdl2:DataField>
</xpdl2:DataFields>
```

Customizing Configurations for Process Editors

You can customize configurations for process editors (to change such things as the available object type, exclusion, and font size).

Studio Analyst Edition: You can see and change the configurations directly in the following configuration file:

- *TIBCO-HOME*\eclipse-platform\bundlepool\1.0\org.eclipse.equinox.p2.touchpoint.eclipse\plugins\com.tibco.xpd.rcp_*n.n.n.nnn*\plugin_customization.ini
- The necessary configurations to exclude object types are included by default in this `ini` file. You can set the value of these configurations (**before starting TIBCO Business Studio**).
- In order for changes to Pools exclusion to be reflected in New Process Wizard templates images, the internal workspace must be recreated. Shut down all running instances of TIBCO Business Studio Analyst Edition and then delete *<user home>*/rcp-workspaces.

Studio BPM Edition: The example configurations are provided in the following file :

- *TIBCO-HOME*\eclipse-platform\bundlepool\1.0\org.eclipse.equinox.p2.touchpoint.eclipse\plugins\com.tibco.xpd.process.editor.branding_*n.n.n.nnn*process_editor_preference_options.txt

The configurations in this file must be **copied** to the product configuration file: *TIBCO-HOME*\eclipse-platform\bundlepool\1.0\org.eclipse.equinox.p2.touchpoint.eclipse\plugins\com.tibco.xpd.branding_*n.n.n.nnn*\plugin_customization.ini.

- Changing the configurations in this `process_editor_preference_options.txt` only has no affect. In order for changes to Pools exclusion to be reflected in New Process Wizard template images, you must create a new workspace on the start-up of TIBCO Business Studio.

Notes:

- Object type exclusions exclude the visibility of the following types of objects:
  - Tool Palette
  - Property sheet type select
  - on diagram context menu type selection
  - on diagram change object type gadget
  - on diagram connect and create gadget

Any pre-defined object that is already of an excluded object type will still be visible, as will its object type in the property sheet (unless/until the object type is changed).

- Pool is a special case - this prevents Pools being displayed in a diagram **except** if there are multiple pools or pools with multiple lanes (in which case the pools and lanes must be made visible). Note that process creation templates images show pools for workspaces that existed prior to changing the configuration to exclude pools.

- Exclusions are per-process-diagram types (Business Process, Pageflow Process and Business Service). For example:

```
Business Process Excluded Object Types ############
# Set to true to exclude object types from the process editor tool palette and type
selection menus, properties and diagram widgets.

# Business Process Task types..
com.tibco.xpd.analyst.resources.xpdl2/BusinessProcess_Exclude_task_none=false
com.tibco.xpd.analyst.resources.xpdl2/BusinessProcess_Exclude_task_user=false
```

```
com.tibco.xpd.analyst.resources.xpdl2/BusinessProcess_Exclude_task_manual=false
com.tibco.xpd.analyst.resources.xpdl2/BusinessProcess_Exclude_task_service=false
com.tibco.xpd.analyst.resources.xpdl2/BusinessProcess_Exclude_task_script=false
com.tibco.xpd.analyst.resources.xpdl2/BusinessProcess_Exclude_task_send=false
```

- The default entry for process editor font sizing preference can be edited. For example, edit the font size of 16 in the Studio for Analysts `ini` file in the example below:

```
# Process diagram editor font size (pitch)
com.tibco.xpd.processwidget/processEditorFontSize=16
```

Visual Characteristics of a Package

This section describes how the visual characteristics of a package/process are created. This would be important for example, if you have your own XPDL that you want to import and display in TIBCO Business Studio.

NodeGraphicsInfo

This XPDL2 element stores graphical information (for example, color, size, and so on) about visual objects, except for connecting lines.

See ConnectorGraphicsInfo.

The following table describes the standard Attributes/Elements of NodeGraphicsInfo.

Attribute/Element	Description
ToolId	For all standard NodeGraphicsInfo elements (for example, activities, pools, or sequence flow), this is unspecified. For special purpose NodeGraphicsInfo elements (those that specify position or size information), use **XPD** for the ToolId. **ToolId** is extended for different purposes (for example, **XPD.BorderEventPosition**). For more information, see Special Purpose NodeGraphicsInfo Elements.
BorderColor, FillColor	String containing three comma-separated numeric values representing Red, Green, Blue values for the color (each between 0 and 255). For example, **255,0,0** (bright red), **0,0,0** (black), and so on.
Height, Width	Height and width of the object in pixels. • Width is never specified for a lane (this is calculated automatically from its content). • Width and Height are never specified for a Pool (these are calculated automatically from its content). • Height is always calculated automatically for diagram notes. If width is present for diagram note, text is wrapped at the specified width; if not present, width is set automatically.

Attribute/Element	Description
Coordinates	**XCoordinate** and **YCoordinate** specify the pixel location of the center of the object, relative to the containing lane or embedded sub-process. **Notes:** • Intermediate events that are attached to a task boundary should be specified as **0.0**. • For text annotations (diagram notes), these coordinates define the center of the the left hand edge.
LaneId	XPDL is not specific about the requirements for various BPMN concepts. Therefore, **LaneId** is dependent on object type. TIBCO Business Studio specifies the **LaneId** value as follows • Activities in a lane - The parent **LaneId**. • Activities in an embedded sub-process - **LaneId** not set. • Artifacts in a lane – The parent **LaneId**. • Artifacts in an embedded sub-process - The **ActivitySetId** for the embedded sub-process.
IsVisible	For lanes only: • false – Lane is closed. • true – Lane is open (default if not present).

Special Purpose NodeGraphicsInfo Elements

In addition to the standard NodeGraphicsInfo elements listed in the previous table, the following special purpose NodeGraphicsInfo element is used when extra graphical information does not fit into the single standard NodeGraphicsInfo element for an object. These are distinguished from the standard NodeGraphicsInfo elements by their extended **ToolId** value.

ToolId	Description
XPD.BorderEventPosition	An Intermediate event on a task boundary is specified as a percentage of distance around the parent task's boundary line (going clockwise from top right corner). This distance is specified in the **Coordinates/XCoordinate** attribute as a floating point number.

ConnectorGraphicsInfo

This XPDL2 element stores graphical information about connection lines such as transition (sequence flows), messageflow and association.

The following table describes the standard attributes/elements of ConnectorGraphicsInfo.

Attribute/Element	Description
ToolId	For all standard ConnectorGraphicsInfo elements this is **XPD.ConnectionInfo**. For special purpose ConnectorGraphicsInfo elements, specify **XPD** as the **ToolId**. **ToolId** is extended for different purposes (for example, **XPD.LabelAnchorPosition**). For more information, see Special Purpose ConnectorGraphicsInfo Elements.
Border Color	Specifies the connection line color as a string containing three comma-separated numeric values representing Red, Green, Blue values for the color (each between 0 and 255). For example, **255,0,0** (bright red), **0,0,0** (black), and so on.

Special Purpose ConnectorGraphicsInfo Elements

In addition to the standard ConnectorGraphicsInfo elements listed in the previous table, the following special purpose ConnectorGraphicsInfo elements are used for extra graphical information that does not fit into the single standard ConnectorGraphicsInfo element for an object. These are distinguished from the standard ConnectorGraphicsInfo elements by their extended **ToolId** value.

ToolId	Description
XPD.StartAnchorPosition	Specifies a fixed position on a source object's boundary for the connection line to start. The actual position is stored in the **Coordinate/XCoordinate** attribute. The value of this depends on the source object type: • **Pool** (Message flows only) - The offset, in pixels, from the left hand edge of the Pool. • **Connection** (Associations only) - The percentage of total length distance from start of the connection line. • Other Objects - The percentage distance around the object's boundary. This is always clockwise, starting from point dependent on the object type: – **Task** – Top right corner. – **Event** – Right hand side (middle). – **Gateway** – Top (middle). – **Data Object** – Bottom left corner. – **Diagram Note** – Top left corner.
XPD.EndAnchorPosition	Specifies a fixed position on target object's boundary for the connection line to end. The actual position is stored in the **Coordinate/XCoordinate** attribute. The value of this is as for the XPD.StartAnchorPosition detailed above.

ToolId	Description
XPD.LabelAnchorPosition	Specifies the anchor position for a connection's label in relation to the connection line itself as two Coordinate elements. • The first Coordinate specifies the anchor point of the label as a percentage distance along connection line (from start of connection) in the **XCoordinate** attribute. • The second Coordinate specifies a horizontal and vertical offset from the anchor position (in the **XCoordinate** and **YCoordinate** attributes).
XPD.LabelSize	**XCoordinate**=Width, **YCoordinate**=Height. If text wrapped at the given width is too large for the height, the height is automatically adjusted on screen.

References Between Elements

This section describes the values that TIBCO Business Studio specifies for various standard XPDL2 cross references between elements.

For top-level elements (objects on a process diagram), TIBCO Business Studio treats the XPDL2 Name attribute as a token name, and uses the XPDL2 Name attribute for all references by name. There is also a label name that is stored in **XPDExt:DisplayName** and used for display purposes only.

Element	Cross Reference Value
Reference Task	A reference task references the target task using the target task's **Activity/Id** attribute.

Element	Cross Reference Value

Intermediate Link Event	The throw event of the link event pair specifies the **Activity/Id** attribute of the catch event (using the **TriggerResultLink/Name** attribute). The process reference is **WorkflowProcess Id** of the catch event's parent process.
	The catch link event specifies its own **TriggerResultLink/Name** attribute as **0**, process is always **-unknown-**.
	For example:
	`<xpdl2:Activity Id="_O1nEsHsoEd2RMpxlT5DsGQ" Name="LinkToCatchLinkEvent" xpdExt:DisplayName="Link To: CatchLinkEvent">`
	`  <xpdl2:Event>`
	`  <xpdl2:IntermediateEvent  Trigger="Link">`
	`  <xpdl2:TriggerResultLink`
	`  CatchThrow="THROW"`
	`  Name="_Q1uiUHsoEd2RMpxlT5DsGQ"/>`
	`  </xpdl2:IntermediateEvent>`
	`  </xpdl2:Event>`
	`.`
	`.`
	`.`
	`</xpdl2:Activity>`
	`<xpdl2:Activity Id="_Q1uiUHsoEd2RMpxlT5DsGQ" Name="CatchLinkEvent" xpdExt:DisplayName="Catch Link Event">`
	`  <xpdl2:Event>`
	`  <xpdl2:IntermediateEvent  Trigger="Link">`
	`  <xpdl2:TriggerResultLink`
	`  CatchThrow="CATCH" Name="0"/>`
	`  </xpdl2:IntermediateEvent>`
	`  </xpdl2:Event>`
	`.`
	`.`
	`.`
	`</xpdl2:Activity>`

Activity Performers (Task Participants)	Activity performers are references to the Ids of participants, performer type data fields, or performer type formal parameters.
Data Fields / Formal Parameters	All references to data fields and formal parameters (except from activity performers), use the name of the field or parameter.
Type Declarations	References are from data field / formal parameter by the type declaration's Id.
Processes	Inter-process references (for example from sub-process call tasks) use the process Id. References to a process in another package use the XPDL filename of the package (without the .xpdl extension).

Schema Extensions

TIBCO Business Studio uses the standard XPDL 2.1 schema with extensions for implementation elements such as invocation style or delayed release type for integration steps.

Schema Extensions for Service Tasks

TIBCO Business Studio uses the standard XPDL 2.1 schema with extensions for certain types of service tasks.

All other types of Service Task (for example, a web services Service Task) conform to the base XPDL 2.1 schema.

Simulation Schema Overview

When you import a Process into TIBCO Business Studio and with Simulation as the destination environment, default simulation parameters are added for Simulation. If you want to over-ride any default simulation parameters, use the simulation schema extensions as described in this section.

You can view HTML documentation for the simulation schema extensions and also actual schema by clicking the following links:

- Simulation HTML
- Simulation XSD

Implementation

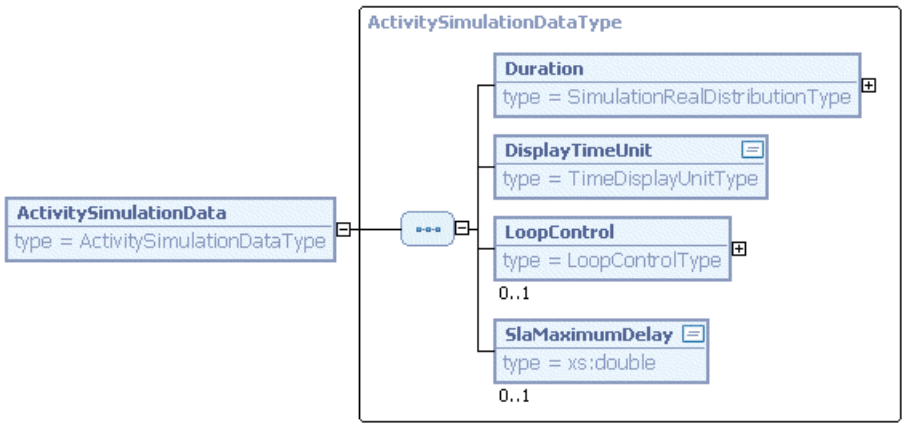
The Activity child **Implementation** and its child (**No**) must be present for simulation (there is no impact on modeling).

For example:

```
<xpdl:Implementation>
  <xpdl:No/>
</xpdl:Implementation>
```

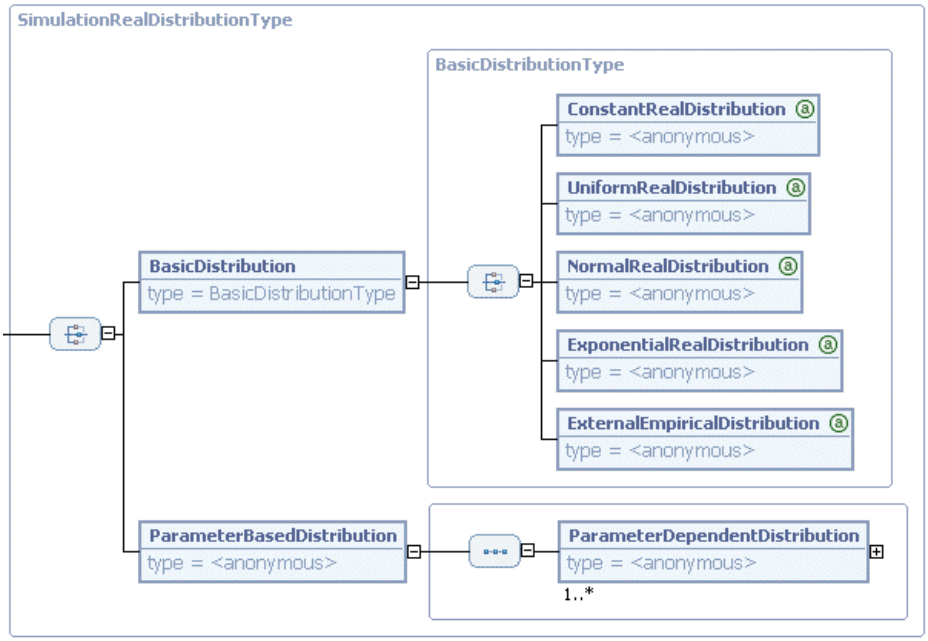
Activity Parameters (ActivitySimulationData)

This section describes the extensions that allow you to specify simulation data for activities:



Specifying Duration Distributions

The following allows you to specify either basic distribution types (where the Activity duration is defined by a mathematical distribution) or a parameter-based distribution (where imported parameters are used):



For basic distributions, you can specify the following types:

Attribute	Example	Notes
ConstantReal Distribution	`<simulation:ConstantRealDistribution ConstantValue="5.0"/>`	Specify a decimal value for **ConstantValue**.
UniformReal Distribution	`<simulation:UniformRealDistribution LowerBorder="2.0" UpperBorder="5.0"/>`	Specify decimal values for **LowerBorder** and **UpperBorder**.
NormalReal Distribution	`<simulation:NormalRealDistribution Mean="5.0" StandardDeviation="2.0"/>`	Specify a decimal value for the **Mean** and **StandardDeviation**.
Exponential RealDistribution	`<simulation:ExponentialRealDistribution Mean="5.0"/>`	Specify a decimal value for **Mean**.

Parameter-based distributions allow you to specify a distribution for each parameter. For example, in the example below:

- First section specifies the distribution for the default case.
- Second section specifies the distribution for new customers (ExistingCustomer=No).
- Third section specifies the distribution for existing customers (ExistingCustomer=Yes

```
<simulation:Duration>
 <simulation:ParameterBasedDistribution>
  <simulation:ParameterDependentDistribution>
   <simulation:BasicDistribution>
    <simulation:NormalRealDistribution Mean="4.25" StandardDeviation="2.436698586202241"/>
   </simulation:BasicDistribution>
   <simulation:Expression>
    <simulation:Default/>
   </simulation:Expression>
  </simulation:ParameterDependentDistribution>
  <simulation:ParameterDependentDistribution>
   <simulation:BasicDistribution>
    <simulation:NormalRealDistribution Mean="7.0" StandardDeviation="0.816496580927726"/>
   </simulation:BasicDistribution>
   <simulation:Expression>
    <simulation:EnumBasedExpression enumValue="No" paramName="ExistingCustomer"/>
   </simulation:Expression>
  </simulation:ParameterDependentDistribution>
  <simulation:ParameterDependentDistribution>
   <simulation:BasicDistribution>
    <simulation:NormalRealDistribution Mean="2.6" StandardDeviation="1.3564659966250536"/>
   </simulation:BasicDistribution>
   <simulation:Expression>
    <simulation:EnumBasedExpression enumValue="Yes" paramName="ExistingCustomer"/>
   </simulation:Expression>
  </simulation:ParameterDependentDistribution>
 </simulation:ParameterBasedDistribution>
</simulation:Duration>
```

In this example, three different distributions are specified, depending on whether the customer is new or existing; all other values are handled by the default.

You must specify a default distribution to handle parameters with values other than the ones you explicitly specify.

Specifying the Time Unit

The time unit specifies the time unit that is used for display purposes in the TIBCO Business Studio user interface. All values for time units (with the exception of SLA information) are stored in the XPDL in minutes. They are then displayed in the user interface in the unit you specify for **TimeDisplayUnit**.

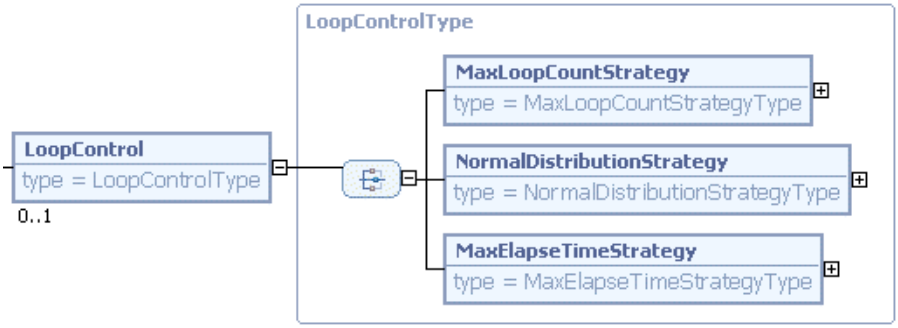
Example

```
<simulation:TimeDisplayUnit>HOUR</simulation:TimeDisplayUnit>
```

Valid values include YEAR, MONTH, DAY, HOUR, MINUTE, and SECOND.

Adding a Looping Control Strategy

A looping control strategy provides the Simulation engine with the necessary mechanism to break out of loops:



MaxLoopCountStrategy

This strategy allows sequence flow to traverse a loop up to the specified maximum number of times.

```
<simulation:LoopControl>
    <simulation:MaxLoopCountStrategy>
      <simulation:DecisionActivity>5</simulation:DecisionActivity>
      <simulation:ToActivity>8</simulation:ToActivity>
      <simulation:MaxLoopCount>5</simulation:MaxLoopCount>
    </simulation:MaxLoopCountStrategy>
</simulation:LoopControl>
```

Attribute	Notes
DecisionActivity	Specifies the Activity that will be used to decide whether to end the loop.
ToActivity	Specifies the Activity to proceed with once the loop is finished.
MaxLoopCount	Specifies the maximum number of times that the loop should be followed.

NormalDistributionStrategy

This strategy allows sequence flow to traverse a loop based on a normal distribution of times (which you specify by providing the mean and standard deviation).

```
<simulation:LoopControl>
    <simulation:NormalDistributionStrategy>
      <simulation:DecisionActivity>5</simulation:DecisionActivity>
      <simulation:ToActivity>8</simulation:ToActivity>
      <simulation:Mean>2.0</simulation:Mean>
      <simulation:StandardDeviation>0.5</simulation:Standard
      Deviation>
    </simulation:NormalDistributionStrategy>
</simulation:LoopControl>
```

Attribute	Notes
DecisionActivity	Specifies the activity that will be used to decide whether to end the loop.
ToActivity	Specifies the activity to proceed with once the loop is finished.

Attribute	Notes
Mean	Specifies the mean used to construct the normal distribution.
StandardDeviation	Specifies the standard deviation used to construct the normal distribution.

MaxElapseTimeStrategy

This strategy allows sequence flow to traverse a loop for an elapsed period of time.

```
<simulation:LoopControl>
 <simulation:MaxElapseTimeStrategy>
   <simulation:DecisionActivity>5</simulation:DecisionActivity>
   <simulation:ToActivity>8</simulation:ToActivity>
   <simulation:DisplayTimeUnit>MINUTE</simulation:DisplayTimeUnit>
   <simulation:MaxElapseTime>10.0</simulation:MaxElapseTime>
 </simulation:MaxElapseTimeStrategy>
</simulation:LoopControl>
```

Attribute	Notes
DecisionActivity	Specifies the activity that will be used to decide whether to end the loop.
ToActivity	Specifies the activity to proceed with once the loop is finished.
DisplayTimeUnit	Specifies the unit in which the elapsed time is measured. Valid values include YEAR, MONTH, DAY, HOUR, MINUTE, and SECOND.
MaxElapseTime	Specifies the elapsed time in which you want the loop to finish.

Adding SLA Information

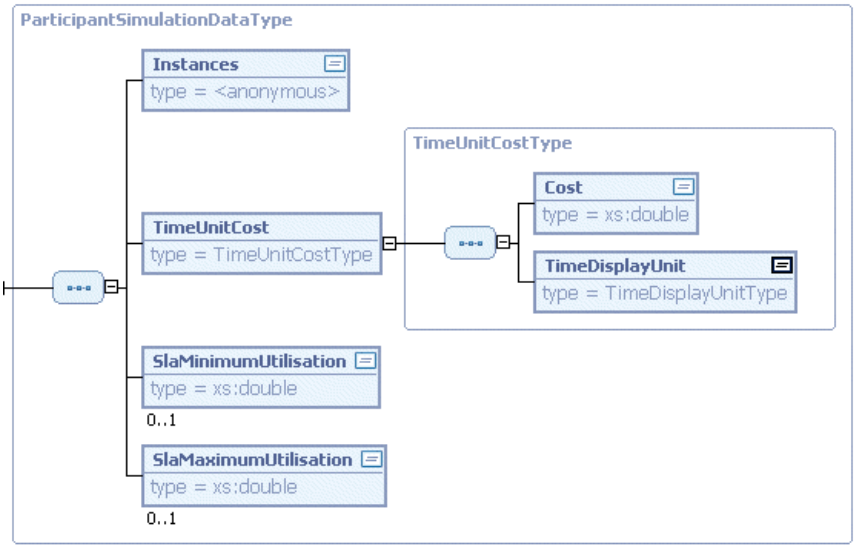
This specifies the maximum amount of delay permissible on the activity:

```
<xpdl:ExtendedAttribute
Name="ActivitySimulationData"><simulation:ActivitySimulationData SlaMaximumDelay="3.0">
```

The unit used for **SlaMaximumDelay** is specified using **TimeDisplayUnit** (see Specifying the Time Unit).

Participant Parameters (ParticipantSimulationData)

This allows you to specify the information about the participant that is necessary for simulation (instances, cost, SLA information and so on).




```
<xpdl:Participants>
  <xpdl:Participant Id="1" Name="Call Center Operator">
    <xpdl:ParticipantType Type="ROLE"/>
    <xpdl:ExtendedAttributes>
      <xpdl:ExtendedAttribute Name="ParticipantSimulationData">
      <simulation:ParticipantSimulationData       SlaMaximumUtilisation="95.0">
         <simulation:Instances>3</simulation:Instances>
         <simulation:TimeUnitCost>
           <simulation:Cost>0.125</simulation:Cost>
           <simulation:TimeDisplayUnit>HOUR
           </simulation:TimeDisplayUnit>
         </simulation:TimeUnitCost>
      </simulation:ParticipantSimulationData>
      </xpdl:ExtendedAttribute>
    </xpdl:ExtendedAttributes>
</xpdl:Participant>
```

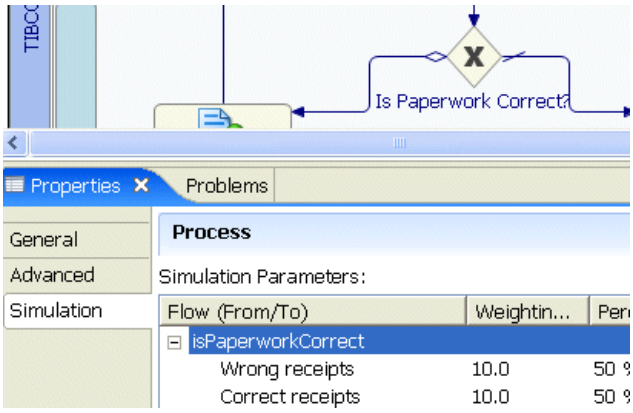
Attribute	Example	Notes
Instances	`<simulation:Instances>1 </simulation:Instances>`	Specifies the number of participants.
TimeUnitCost	`<simulation:TimeUnitCost> <simulation:Cost>0.8333333333333334 </simulation:Cost> <simulation:TimeDisplayUnit>HOUR </simulation:TimeDisplayUnit> </simulation:TimeUnitCost>`	Specifies the cost and time unit used for the participant in simulation.
SlaMinimumUtilisation	`<xpdl:ExtendedAttribute Name="ParticipantSimulationData"> <simulation:ParticipantSimulationData SlaMinimumUtilisation="65.0" SlaMaximumUtilisation="85.0">`	Specifies the minimum utilization percent for the participant. This is displayed visually when the process is simulated.

Attribute	Example	Notes
SlaMaximumUtilisation	see **SlaMinimumUtilisation** example	Specifies the maximum utilization percent for the participant. This is displayed visually when the process is simulated.

Sequence Flow Splits (SplitSimulationData)

This specifies the parameter that is used to distribute sequence flow through a split.

For example:

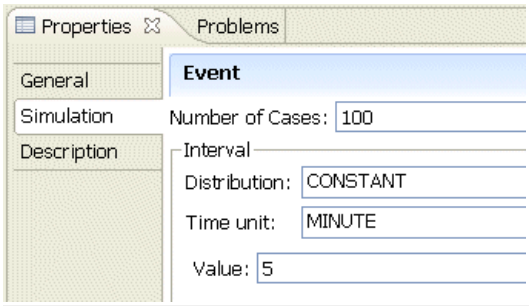


SplitSimulationData specifies the parameter associated with the gateway (in this case **isPaperworkCorrect**):

```
<xpdl:ExtendedAttribute Name="SplitSimulationData"><simulation:SplitSimulationData>
      <simulation:ParameterDeterminedSplit>true
      </simulation:ParameterDeterminedSplit>
    <simulation:SplitParameter ParameterId="isPaperworkCorrect"/>
</simulation:SplitSimulationData></xpdl:ExtendedAttribute>
```

Simulation Start Parameters (StartSimulationData)

This provides the start parameters needed for simulation such as the distribution and number of cases. These are found on the Properties view of the start event:

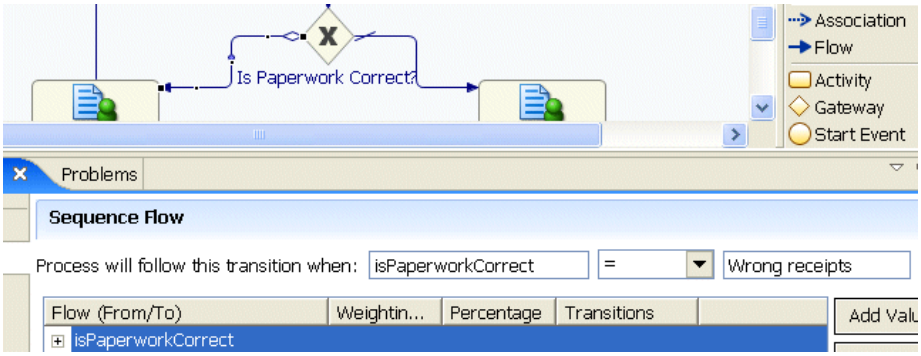



```
<xpdl:ExtendedAttribute Name="StartSimulationData">
<simulation:StartSimulationData NumberOfCases="100">
    <simulation:Duration>
      <simulation:ConstantRealDistribution ConstantValue="5.0"/>
    </simulation:Duration>
    <simulation:DisplayTimeUnit>MINUTE
    </simulation:DisplayTimeUnit>
</simulation:StartSimulationData></xpdl:ExtendedAttribute>
```

Attribute	Example	Notes
NumberOfCases	`<simulation:StartSimulationData NumberOfCases="100">`	Specifies the number of cases for simulation.
Duration	`<simulation:Duration> <simulation:BasicDistribution> <simulation:NormalRealDistribution Mean="5.0" StandardDeviation="2.0"/> </simulation:BasicDistribution> </simulation:Duration>`	Specifies the distribution for case starts as one of the following: • **ConstantRealDistribution** (Specify a decimal value for **ConstantValue**). • **UniformRealDistribution** (Specify decimal values for **LowerBorder** and **UpperBorder**). • **NormalRealDistribution** (Specify a decimal value for the **Mean** and **StandardDeviation**). • **ExponentialRealDistribution** (Specify a decimal value for the **Mean**).

Sequence Flow Parameters (TransitionSimulationData)

This allows you to specify an expression that is evaluated to determine whether a sequence flow is traversed. In TIBCO Business Studio, this is specified as a rule:



In this case, the sequence flow highlighted will only be traversed if the parameter **isPaperworkCorrect** is equal to **Wrong receipts**. This is specified as follows:

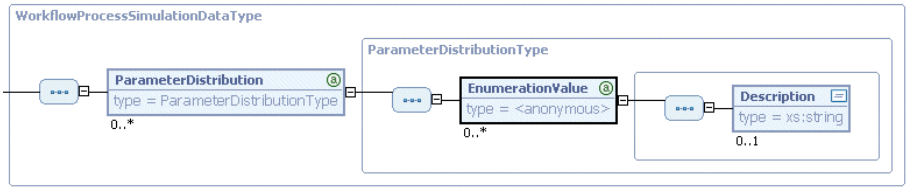
```
<xpdl:ExtendedAttributeName="TransitionSimulationData">
<simulation:TransitionSimulationData>
    <simulation:ParameterDeterminedCondition>true
    </simulation:ParameterDeterminedCondition>
    <simulation:StructuredCondition>
      <simulation:ParameterId>isPaperworkCorrect
      </simulation:ParameterId>
      <simulation:Operator>=</simulation:Operator>
      <simulation:ParameterValue>Wrong receipts
      </simulation:ParameterValue>
    </simulation:StructuredCondition>
</simulation:TransitionSimulationData></xpdl:ExtendedAttribute>
```

This specifies that the sequence flow will only be traversed when the parameter **isPaperworkCorrect** has the value **Wrong receipts**.

Attribute	Example	Notes
ParameterDetermined Condition	`<simulation:ParameterDeterminedCondition>true</simulation:ParameterDeterminedCondition>`	Specifies whether a parameter will be evaluated to determine flow through the sequence flow.
StructuredCondition	`<simulation:StructuredCondition> <simulation:ParameterId>isPaperwork Correct </simulation:ParameterId> <simulation:Operator>=</ simulation: Operator> <simulation:ParameterValue>Wrong receipts </simulation:ParameterValue> </simulation:StructuredCondition>`	Specifies the condition that is evaluated to allow the sequence flow to be traversed. The condition is made up of: • ParameterId • Operator (>, =, >=, <, < =) • ParameterValue

Parameter Distribution (WorkflowProcessSimulationData)

Allows you to define simulation parameters and distributions. Simulation parameters are used in splits and in parameter-based distributions for activities.




```
<xpdl:ExtendedAttributeName="WorkflowProcessSimulationData">
<simulation:WorkflowProcessSimulationData>
    <simulation:ParameterDistributionParameterId=
    "ExistingCustomer">
         <simulation:EnumerationValue Value="Yes"          WeightingFactor="10.0"/>
         <simulation:EnumerationValue Value="No"          WeightingFactor="10.0"/>
    </simulation:ParameterDistribution>
</simulation:WorkflowProcessSimulationData>
</xpdl:ExtendedAttribute>
```

Attribute	Example	Notes
ParameterDistributionId	`<simulation:ParameterDistributionParameterId="ExistingCustomer">`	Specifies the parameter that will be evaluated to determine flow through the split.
EnumerationValue	`<simulation:EnumerationValue Value="Yes" WeightingFactor="10.0"/>`	Specifies the possible values for the parameter.

Attribute	Example	Notes
WeightingFactor	`See EnumerationValue example.`	Specifies the weighting given to each transition in the split. For example, if each transition has the same weighting (10 in the previous example), the split will be 50:50.

Javadoc Locations

Javadoc documentation is available as part of TIBCO Business Studio for the core and BOM features.

For the core feature the index file of the Javadoc can be found here:

docs.com.tibco.xpd.core/html/reference/javadoc/index.html

For the BOM feature the index file of the Javadoc can be found here:

docs.com.tibco.xpd.bom/html/reference/javadoc/index.html

Creating an XSLT-Based Import Export Wizard

This section describes how to create either an import wizard for a file format that you want to import into TIBCO Business Studio or an export wizard for exporting a TIBCO Business Studio package into another file format.

Once you have written the XSLT that will be used either to import your files into TIBCO Business Studio or to export from TIBCO Business Studio into another format, you need to create a wizard that will allow users to perform the export or import. The **New Import/Export Wizard Plug-in Generator Wizard** allows you to do this. Using these, you can create a plug-in for your product with all the required JAR files for exporting from and importing to TIBCO Business Studio.

- This section describes import and export wizards for mappings implemented in XSLT. You can however use the Eclipse extension points and implement mappings in Java although this is beyond the scope of this document.
- Use the `xpdl2html.xsl` XSLT file (located in the **com.tibco.xpd.procdoc** plug-in) as a starting point for creating your own XSLT for the wizard.

  The `xpdl2html.xsl` XSLT file contains Java extension functions that are internal and unavailable to customized exports. The resolution is to remove all such extension functions (search for 'java:'). The resulting XSLT will still be a reasonable starting point since the primary use of the extension functions is to support localisation of the text in the output documentation, and this is just an example to help you get started.

Installation of an Export/Import Plug-in into Post TIBCO Business Studio 3.6.0

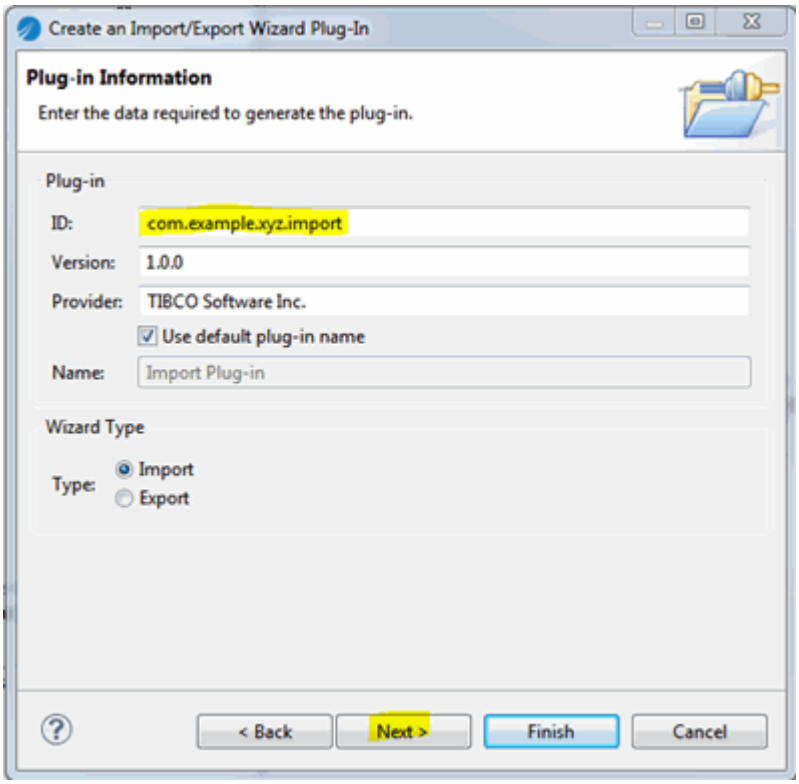
In order to create a custom import/export plug-in and enable it for installation into TIBCO Business Studio you must first create and export the plug-in, create a feature for this plug-in, and finally create an Eclipse P2 Repository containing the feature.

From your P2 Repository your custom import/export feature can be installed into one or more TIBCO Business Studio installations. The following tasks describe how this is achieved in detail.

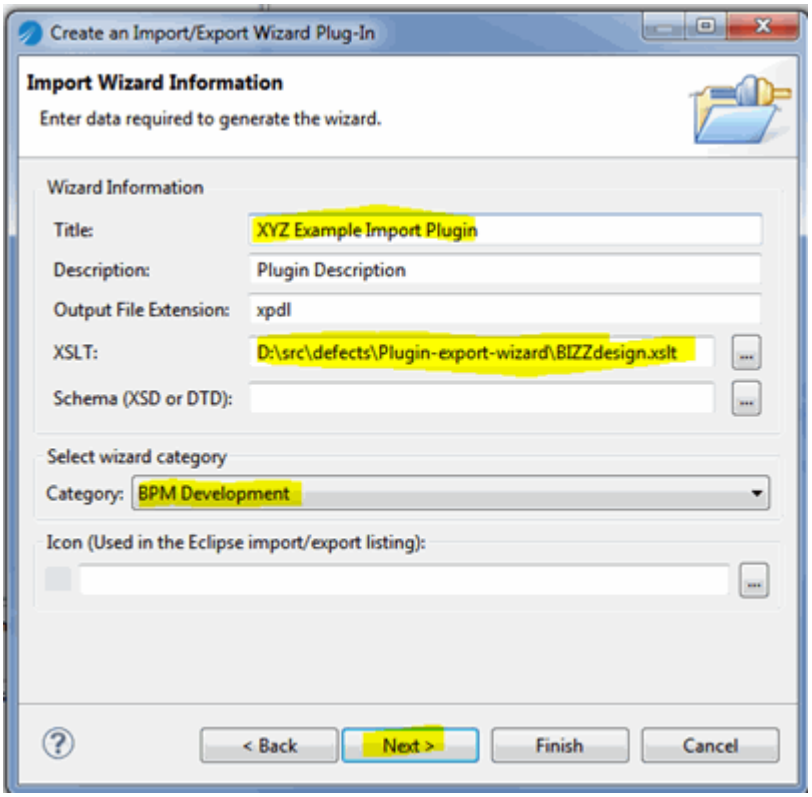
Creating Import/Export Plug-in

Procedure

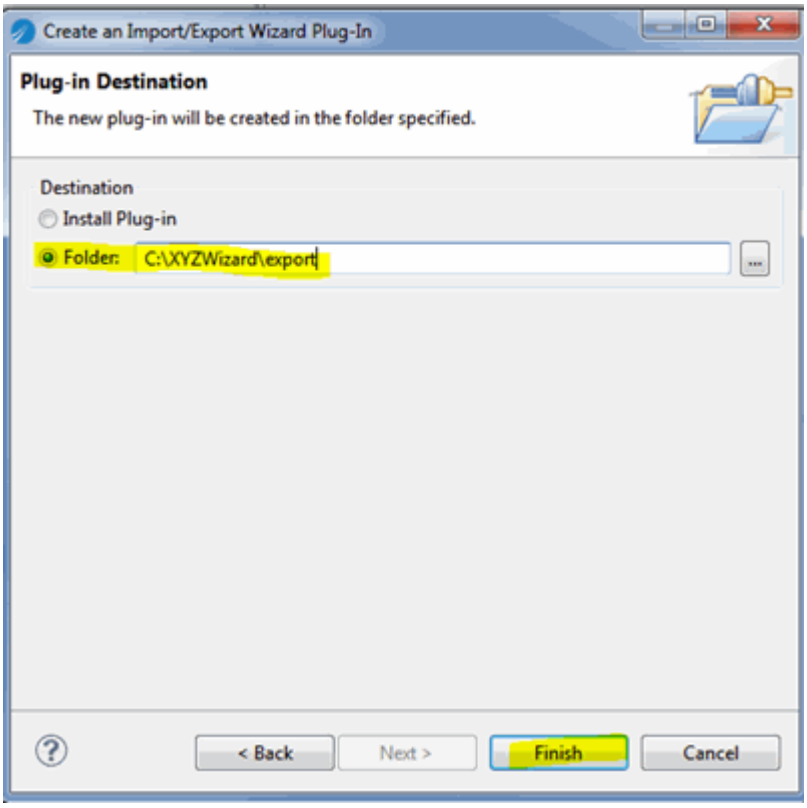
1. Click **File** > **New** > **Other...** and then select **New Import/Export Plug-in Generator Wizard** from the wizard list.
2. Fill in the details on the **Plug-in Information** page (**com.example.xyz.import** in the example below) and click **Next**.



3. Fill in the **Import Wizard Information** page and click **Next** (also provide the schema if necessary).



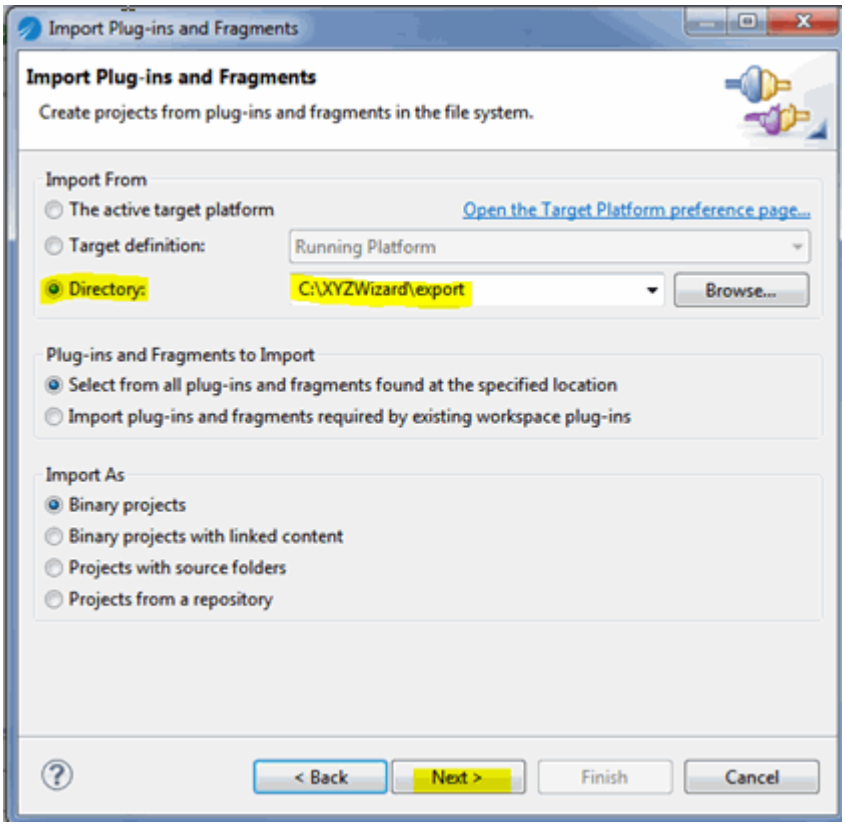
4. Select special folder(s) to filter the content of what can be browsed as a destination folder of the import wizard. You can also use the Set file extensions filter... field to filter on specific extensions in the source files browser of the import wizard (all files will be shown if it is empty). Click **Next**.
5. Click **Folder** radio button to specify a folder to export the new plug-in and then click **Finish**.



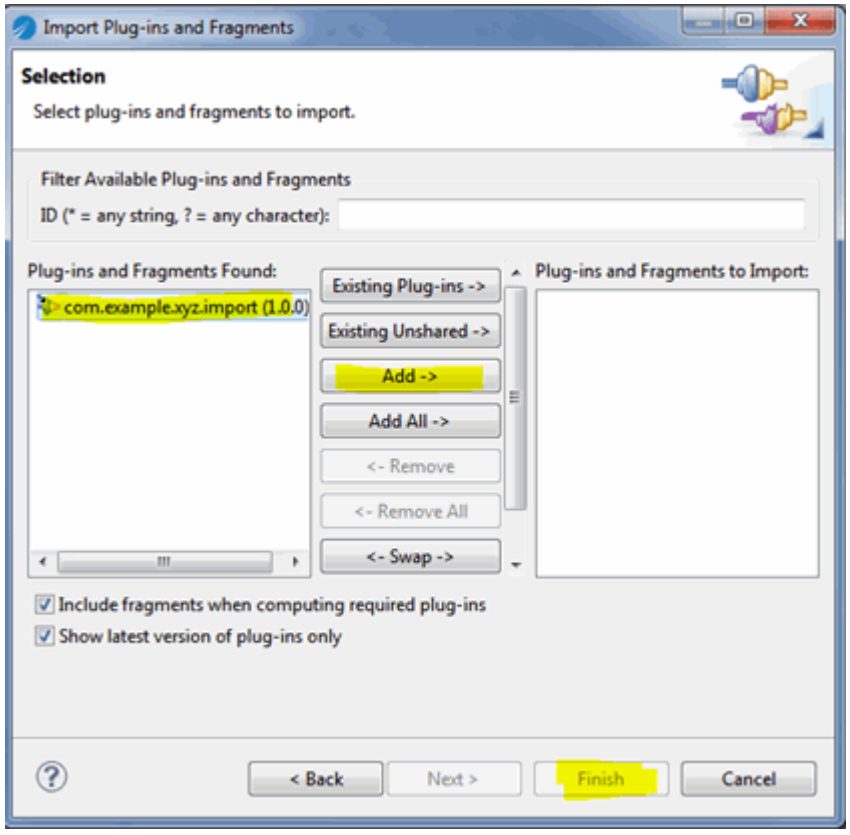
Importing Plug-in into the Workspace

Procedure

1. Click **File** > **Import**, select the **Plug-ins and Fragments** wizard and click **Next**.
2. Under Import From, click **Directory** and specify the folder you used to export the plug-in into step 5 in Creating Import/Export Plug-in and click **Next**.



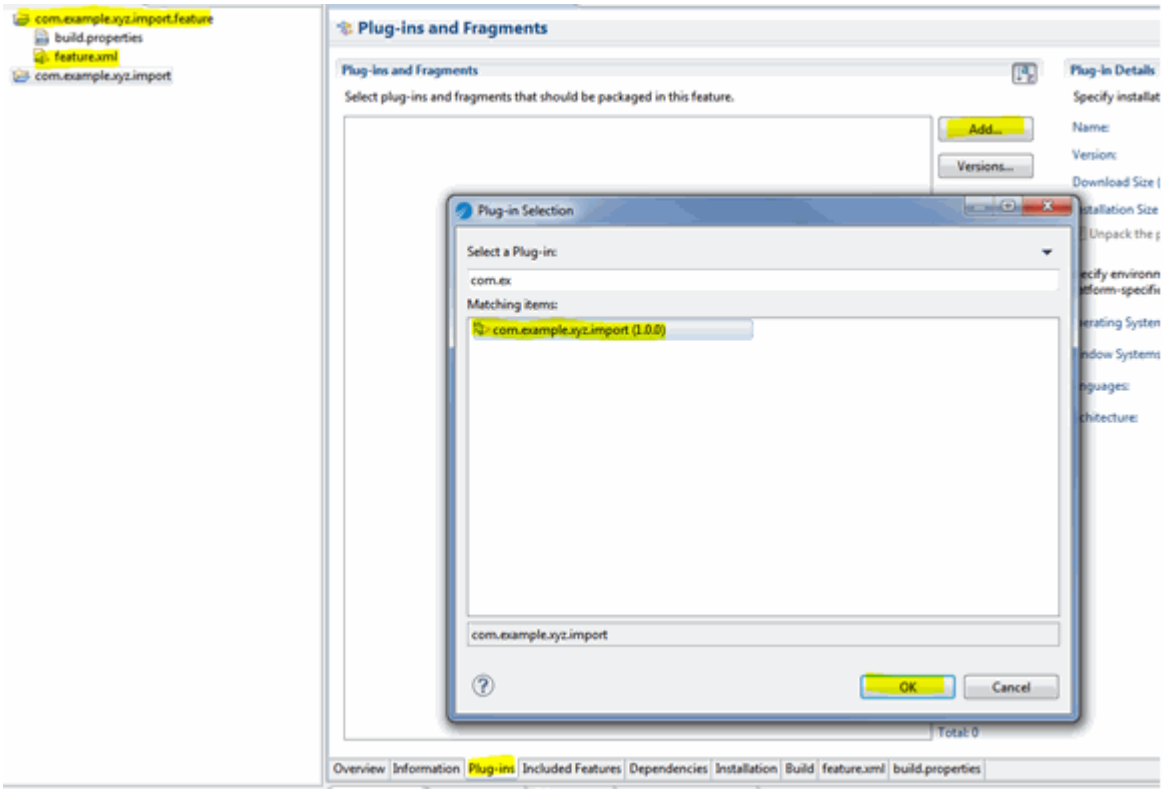
3. Select the import plug-in in the list on the left and click **Add** to add it to the list on the right. Click **Finish**.



Creating a Feature Project

Procedure

1. Click **File** > **New** > **Other...**, select the **Feature Project** in the wizard and click **Next**.
2. Fill in the **Feature Properties** page specifying **Project name** and **Feature name** (for example **com.example.xyz.import.feature** and **XYZ Example Import Feature**) and click **Finish**.
3. Open the **com.example.xyz.import.feature/feature.xml** file in the feature editor, click the Plug-ins tab and add **com.example.xyz.import** plug-in (and then save the file).

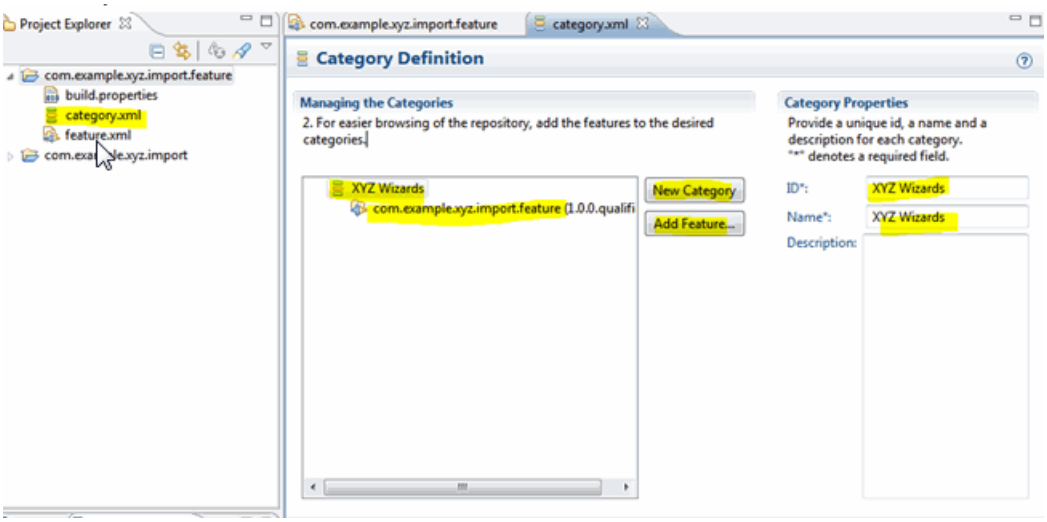


Creating Categories (Optional)

Procedure

1. Create a category definition for the feature:
   a) Select the feature project and click **New** > **Other…**.
   b) Select **Category Definition** wizard.
   c) Click**Next**.
   d) Click **Finish**

   This step should create a `category.xml` file in your feature project and open it in the category definition editor.
2. In the category editor add a new category and fill in its **ID** and **Name** properties (for example you can use "XYZ Wizards" for both), then select the new category and click **Add Feature…**. Select **com.example.xyz.import.feature** in the dialog window and finally save the file.

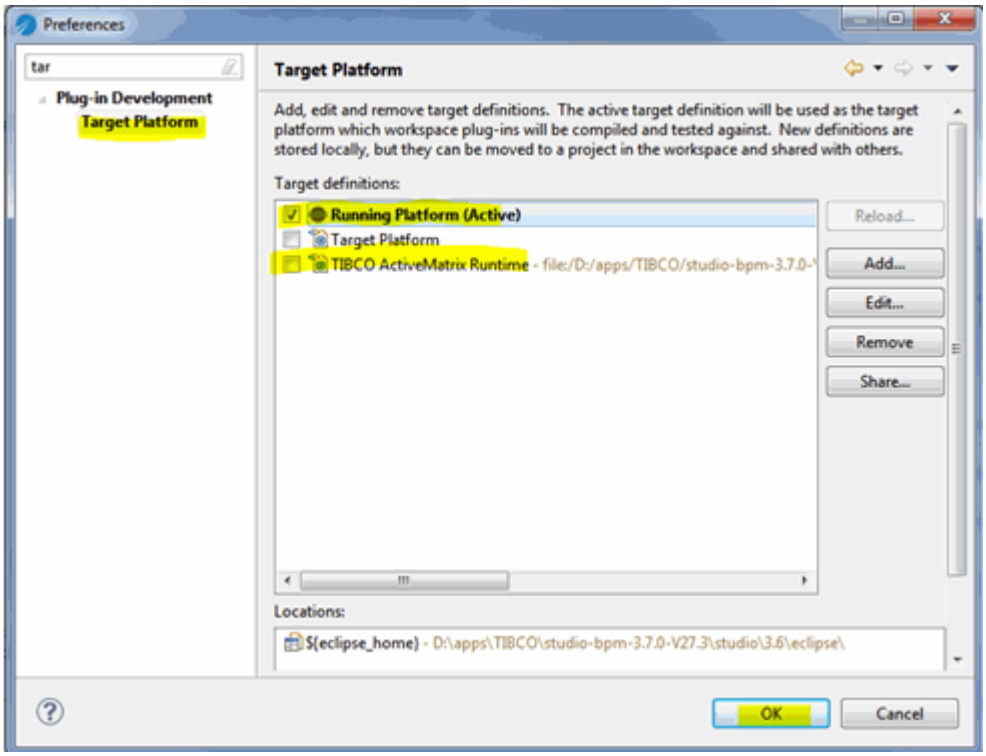


Switching Target Platform to the “Running Platform”

Procedure

- Click **Window** > **Preferences…**, select the **Target Platform** page and switch target platform from **TIBCO Active Matrix Runtime** to **Running Platform** and then click **OK**. This step is necessary to be able to export the plug-in as it requires dependencies which are not present in the TIBCO Active Matrix Runtime.

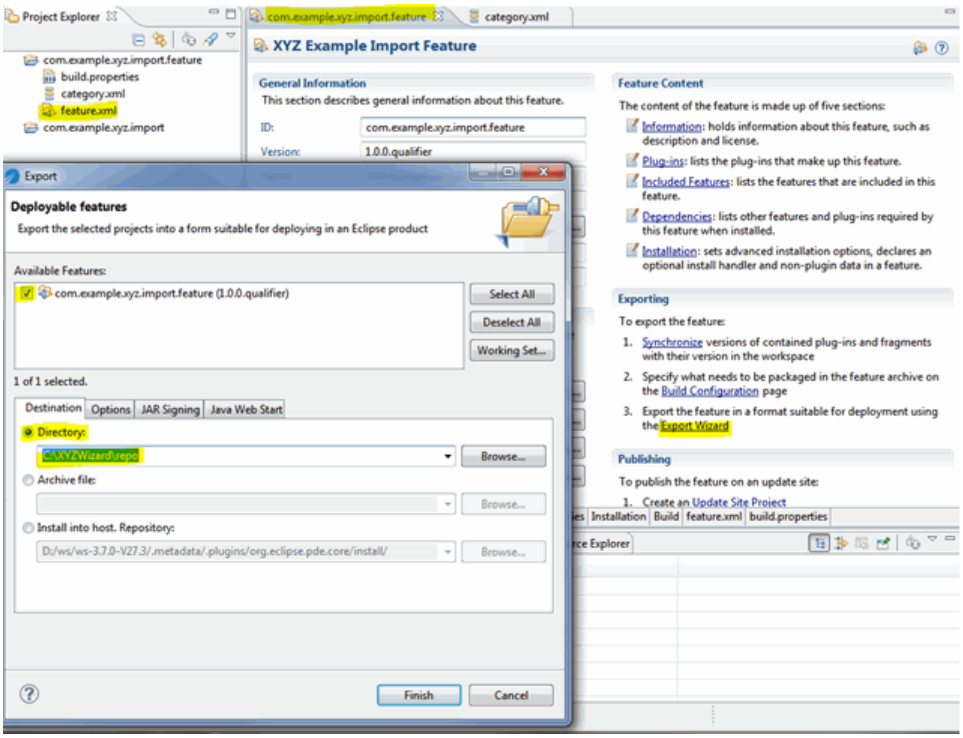
  Please switch back target platform after you have finished exporting the plug-in to be able to export DAAs.



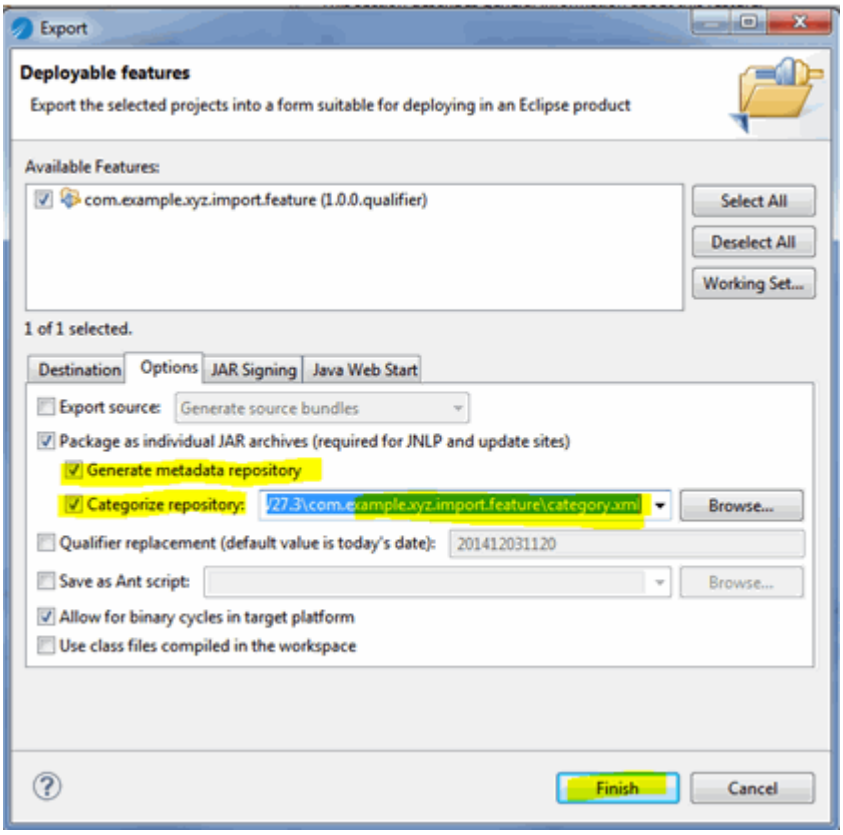
Exporting P2 Repository

Procedure

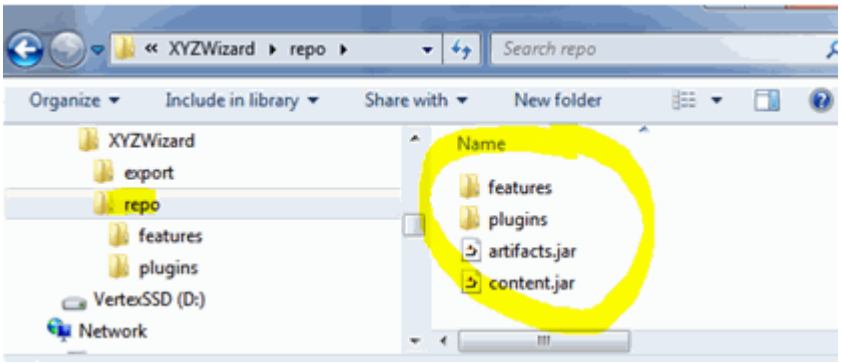
1. Open `feature.xml` in the feature editor and select the **Overview** tab and select the **Export Wizard** link in the **Exporting** section.
2. On the Export wizard's **Destination** tab click **Directory** and specify a folder for the export of the feature (the folder will contain a P2 repository).For example use the `C:\XYZWizard\repo` folder.



3. On the **Options** tab select both the **Generate metadata repository** and **Categorize repository** checkbox. Browse for the `category.xml` file (if you have done the steps in Creating Categories (Optional)) and click **Finish**.



As a result you should be able to see that the P2 repository containing your feature (and plug-in) has been created. (P2 repository is a folder containing installable artefacts in the `features` and `plugins` folders and metadata in the `artifacts.jar` and `content.jar` files.)



Switching the Target Platform Back to TIBCO ActiveMatrix Runtime

Procedure

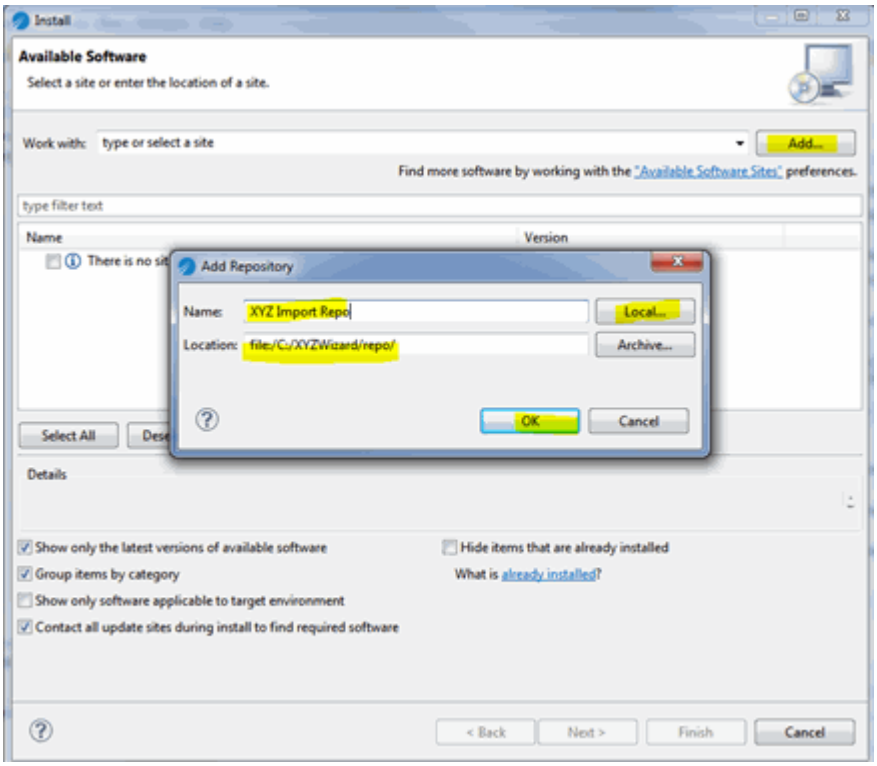
- You **must** switch the target platform back to **TIBCO Active Matrix Runtime**. If the target platform is no longer present, follow the steps in Recreating TIBCO Active Matrix Runtime Target Platform Definition (If it Disappears).

Installing the Feature from the Repository

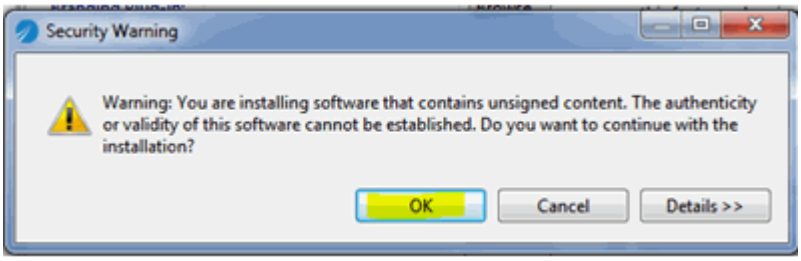
You can now install your feature into multiple TIBCO Business Studio installations on different machines.

Procedure

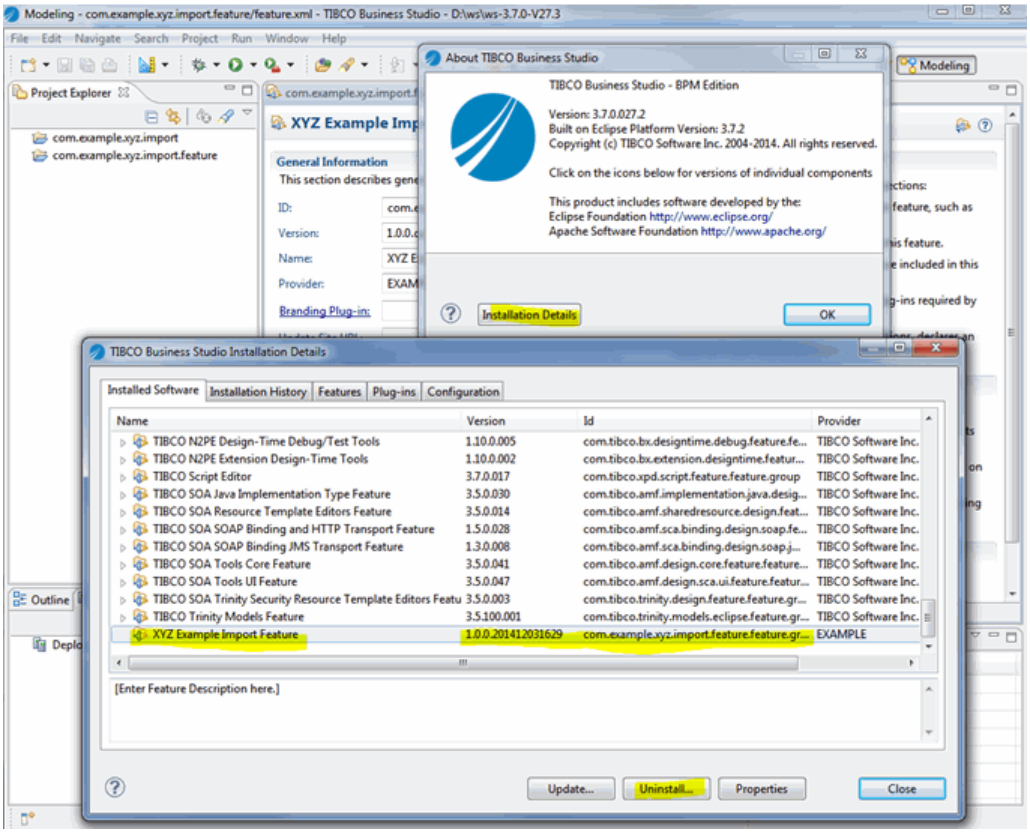
1. Click **Help** > **Install New Software…** and press **Add…** to add a new repository. Click **Local…** and choose `C:\XYZWizard\repo` folder, then set the name of the repository (for example, `XYZ Import Repo`). Click **OK** .



2. Select your feature to install and click **Next** and you see the Install details. (If you haven't specified a category you may need to clear the **Group items by category** check box to be able to see and select the feature to install.)
3. Click **Next** to accept the licenses (this can be filled in in the feature editor) and click **Finish**. Accept the warning about unsigned content by clicking **OK**, and restart Eclipse at the end of the installation.



4. After restarting Eclipse you can confirm that your feature has been installed successfully by clicking **Help** > **About TIBCO Business Studio** and clicking **Installation Details**.



What to do next

The feature can be uninstalled by clicking **Uninstall…**.

You **must** switch back to the **TIBCO ActiveMatrix Runtime** target platform when you have finished creating your P2 Repository. See Recreating TIBCO Active Matrix Runtime Target Platform Definition for steps to take if your TIBCO ActiveMatrix Runtime target platform is missing.

Recreating TIBCO Active Matrix Runtime Target Platform Definition (If it Disappears)

If TIBCO Business Studio is re-started at anytime when the target platform is switched to the running platform (see Switching Target Platform to the “Running Platform”) then it is possible that the TIBCO ActiveMatrix Runtime target platform will no longer be available for selection. In this case you can simply create a new TIBCO Business Studio workspace and or re-create the target platform using the following steps.

Procedure

1. From **Preferences** > **Target Platform** click **Add…** to create a new **Target Definition** and on the first wizard page choose **Nothing: Start with an empty target definition** and click **Next**.
2. On the **Target Content** page, type `TIBCO ActiveMatrix Runtime` into the **Name** field and click **Add…**.
3. On the **Add Content** page select the **Dictionary** option and click **Next**, fill in the Location field with `${eclipse_home}../../../components/shared/1.0.0` and click **Finish**.
4. Click **Finish** to end **New Target Definition** wizard.
5. Select the newly created **TIBCO ActiveMatrix Runtime** target definition and accept with **OK** button.

Deployment Framework

By default, TIBCO Business Studio includes deployment to the iProcess engine. This section describes how to use the deployment framework to create a server so you can deploy your own artifact (Module) on a remote server other than iProcess.

You should be proficient in Eclipse and Java development before using the deployment framework. Recommended reading: *Eclipse: Building Commercial-Quality Plug-ins*, by Eric Clayberg and Dan Rubel.

The deployment framework allows you to deploy a resource (represented in TIBCO Business Studio as a **Module**) on a local or remote system (represented in TIBCO Business Studio as a **Server**). The Module can be a resource or set of resources, but it must be located using a Uniform Resource Locator (URL).

The deployment framework assumes that the physical machine referred to by the Server is running; you cannot start and stop the Server from within TIBCO Business Studio. However, you can connect and disconnect from the Server. Once connected, you can manage Modules and other objects on the Server by interrogating their states and performing operations on them.

The typical way deployment is used is as follows:

- The user creates a Server within TIBCO Business Studio that contains all the necessary connection parameters and details.
- The user connects to the running Server, deploys Modules, interrogates server objects and performs operations on them.
- The user disconnects from the Server.

Deployment Repository

The deployment framework also supports the concept of a **deployment repository**. Modules that are going to be deployed are placed in the repository where the Server can obtain instances of the Modules to deploy.

Using the repository allows the separation of deployment into two distinct phases:

- Providing the Module to the Server (for example, by sending them to a location known to the server).
- Actual deployment

Deployment repositories only handle the first phase of deployment (providing the Module).

Deployment Policy

The deployment framework also handles the **deployment policy**.

Modules are stored in the Eclipse workspace, which means that they can have the following deployment policies:

- Deploy on request - the Module is only deployed (or redeployed) when the user explicitly chooses to do so.
- Deploy on save - the Module is deployed (or redeployed) whenever the Package is saved.

Implementing Deployment

This section describes the steps that you must follow to implement deployment.

The deployment framework provides the following:

- definition of the server type
- abstract connection
- server element structure.

TIBCO Business Studio provides extension points (some of which are optional) that you can implement to do the following:

- define the server
- specify the method of connection to the server
- define how the server elements will be retrieved from servers
- detail the operations and states that are possible from each operation.

Define the Module

The artifacts that you want to deploy are managed as Modules in TIBCO Business Studio. You also need to define how the Modules are created.

Define the Management Operations

When a Module is deployed on the server, you can use the TIBCO Business Studio user interface to perform operations on the deployed Modules. You must define these operations.

Defining the Server

The **com.tibco.xpd.deploy.core.ServerType** extension point allows you to define the type of server on which you want to deploy. In your extension, you must provide a reference to a class that implements **com.tibco.xpd.deploy.model.extension.ConnectionFactory**, and optionally all necessary server configuration parameters. **ConnectionFactory** has a createConnection method that takes a Server object and should return a connection specific to your server.

The connection to the server is represented by a class that implements the **com.tibco.xpd.deploy.model.extension.Connection** interface. For example:

```
public interface Connection extends IAdaptable {
    void connect();
    void disconnect();
    boolean isConnected();
    void refreshServerContent();
    Object deployModule(String url);
    Object performServerElementOperation(ServerElement        serverElement,
           Operation operation);
    Server getServer();
}
```

The **Connection** interface has a validateModule() method that validates a module for deployment on the server. This method must be implemented. If true is returned, the module is deployed. If false is returned, the module is not deployed. Deployment extensions developed against TIBCO Business Studio Version 2.0 must be recompiled against a later version. This method must be implemented, but can return true to continue the existing behavior.

Connecting to a Server

Connection holds a reference to a server which you can obtain using the **getServer()** method. The **Connect** method is responsible for connecting to a Server. All necessary parameters needed to obtain connection can usually be taken from the server configuration, which is initialized when the server is created. The available configuration parameters are defined in the **serverTypes** extension point. The values for these parameters are provided by the user when creating an instance of the Server using the **New Server Wizard**.

When the connection with the server is established and valid, the **isConnected()** method should return **true**; otherwise it returns **false**.

When the connection is no longer needed, you can close the connection by invoking the **disconnect()** method.

Defining a Runtime (Optional)

A Runtime is configuration data that is shared amongst many server types. For example, if you use a client-side management system such as a JMX console to manage many servers, this server type could use common configuration details. This is implemented using the extension point **com.tibco.xpd.deploy.core.runtimeTypes** and by providing values for the necessary runtime parameters.

If a server type has an associated runtime, it should be reflected in the **serverTypes** extension by providing the associated **RuntimeType** sub element with the identifier of the runtime type extension.

Define the Possible Server Elements

Next, you must define the possible server elements and the method for obtaining them.

You must do the following:

- define the hierarchy and way of displaying the elements in the Project Explorer
- you must create the structure such that it conforms to the Composite Design Pattern.

When the Framework needs to refresh, the method **refreshServerContent** retrieves the structure of objects on the server from which it builds the tree of server elements from the model.

Define States for Elements

You must define the possible states in which an element can be. For example, an iProcess procedure can be Released, Unreleased, Withdrawn, and so on.

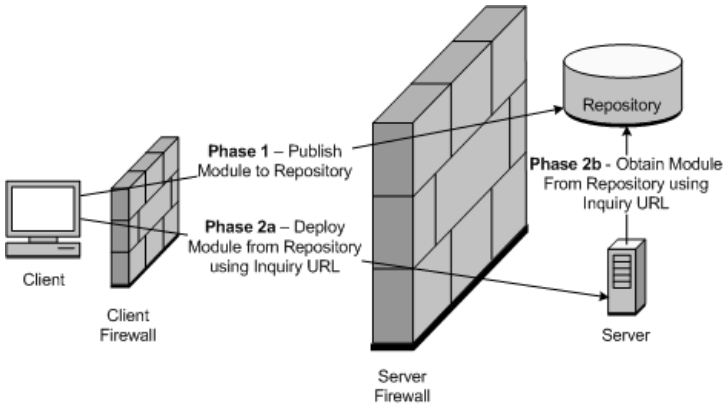
In this process you define the possible states for a given element type. For each operation, you also define the setTo state (the state that logically follows the success of the operation). The method **getPossibleOperations** defines the legal progression of states. For more information see Implementing Operations for ServerElements.

The operation names and states that you choose will be displayed in the user interface.

Configuring the Repository

The Repository is where a server can obtain Modules which are going to be deployed.

The following diagram shows the phases of deployment based on the Java Business Integration (JBI) example:



- **Phase 1** - The Module is published to the Repository and the inquiry URL is evaluated. The inquiry URL is an absolute or a relative URL of the Module in the Repository which is known and accessible from the server.
- **Phase 2a, 2b** -The **deploy module** operation is invoked on the server. The parameter of the deployment operation is the inquiry URL of the Module. The **deployModule(String inquiryUrl)** method which is a part of the **Connection** interface is responsible for this step.

Repository Types

Each server type must be associated with at least one repository type.

There are two pre-defined repository types:

- **Workspace** - In this case, the publish action (Phase 1) does nothing, and the inquiry URL is the same as the URL of the Module in the workspace. This repository type is **default**, and it works well if the server is on the same machine as the client and the server can deploy Modules directly from the Eclipse workspace.
- **Local folder** - The repository will be located in a local folder or equivalent (for example mapped network folder, or WebDAV mapped folder). This repository type requires two additional configuration parameters:
  - **Publishing folder** - a path to the repository folder.
  - **Inquiry URL prefix** - a prefix of the enquiry URL that can be resolved on the server (the URL server will use this to obtain the module).

If you need to define another type of repository, implement the **com.tibco.xpd.deploy.core.repositoryTypes** extension point and reference the repository from the **serverType** extension. You also must provide the implementation of the **com.tibco.xpd.deploy.model.extension.RepositoryPublisher** interface for the newly defined repository type and provide any additional configuration parameters that are needed:

```
public interface RepositoryPublisher {
     public void publish(RepositoryConfig config, File file);
     public URL getInquiryUrl(RepositoryConfig config, File file);
}
```

The **publish(RepositoryConfig config, File file)** method is responsible for publishing Modules to the repository and will be invoked before deployment. The **config** parameter references all repository configuration parameters and their values (that were provided when the server was created). The second method: **getInquiryUrl(RepositoryConfig config, File file)** is for obtaining the inquiry URL corresponding to the file parameter.

If the repository type needs parameters, the values for which should be provided by the user, you should implement a repository configuration wizard page. To do this, extend the **com.tibco.xpd.deploy.ui.repositoryConfigWizardPage** extension. This extension requires the implementation of the **com.tibco.xpd.deploy.ui.wizards.repository.RepositoryConfigWizardPage** interface that defines the additional wizard page with configuration details:

```
public interface RepositoryConfigWizardPage extends IWizardPage {
    void init(RepositoryType type, RepositoryConfig config);
    void transferStateToConfig();
}
```

The **init(RepositoryType type, RepositoryConfig config)** method will be invoked before the page is created. This provides the **RepositoryConfig** reference that can be interrogated for configuration parameters and filled by the user accordingly. The configuration state can also be cached in controls and transferred to the configuration object just before finish. In this case, the appropriate code to transfer the state to the configuration has to be put to the **transferStateToConfig()** method.

Define the Deployment Wizard

You need to define a wizard that allows users to deploy your Module. The key to this is that your artifact must be capable of being represented as a URL. Your deployment wizard must conform to the Eclipse iDeployWizard. This requires the implementation of **getModulesURL**.

In **ui.deployWizards**, you make the association between the server type id and the wizard.

When you deploy on the server, the extension point registry is searched, and the available wizards for the server type that you are deploying are displayed. The user then chooses a wizard.

Worked Example - Deployment to a WebDAV Server

This example shows how to use the TIBCO Business Studio Deployment Framework to create a plug-in for the deployment of a workspace file to a WebDAV (Web-based Distributed Authoring and Versioning) server. It also describes how to retrieve server objects, information about them, and how to implement operations performed on these objects.

WebDAV is an extension of the HTTP protocol designed to facilitate editing of web resources. In this tutorial, we will create a deployment system based on a WebDAV folder (also called a collection in WebDAV terminology). The TIBCO Business Studio Deployment module will be any workspace file. During deployment, the file will be put to the selected WebDAV folder which in our example we will call “site”. We will also display content of the site underneath the server and implement a delete operation which you can use to remove deployed files.

Prerequisites before you follow the Example

This section lists the tasks that you must complete before you can follow the example.

Creating the Server

Create or obtain access to a WebDAV compatible server. Open source alternatives such as Apache Web Server WebDAV module are available as well as WebDAV support in commercial web server products. This tutorial uses Apache Slide, which is based on the Tomcat application server.

To install Apache Slide do the following:

Procedure

1. Download the binary Tomcat bundle from:

   http://jakarta.apache.org/slide/download.html
2. Unpack the zipped package.
3. Set the JAVA_HOME environment variable to root directory of Java SDK and run startup.bat (startup.sh on Unix systems). For detailed installation instructions, see http://jakarta.apache.org/slide/installation.html.

Result

By default the server runs on port 8080, the default path to the WebDAV root is /slide, and the username and password are “root”.

To test if the installation was successful, enter http://localhost:8080/slide in your browser. After providing the correct username and password, a simple directory structure should be displayed.

Set the Target Platform

Set the target platform as follows:

Procedure

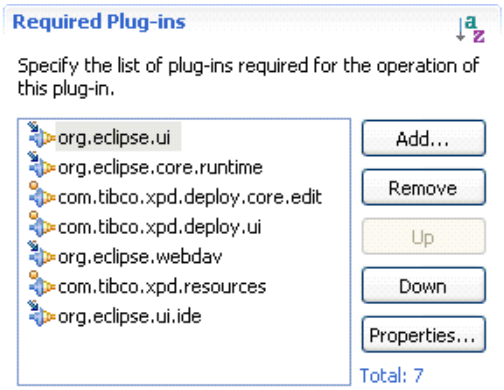
1. Install the latest release of TIBCO Business Studio.
2. From the Eclipse development environment, select WindowPreferences, then Plug-in DevelopmentTarget Platform.
3. Browse for the location where you installed TIBCO Business Studio, and click **Reload** to load the target platform.

4. When the target platform is finished loading, click OK.

Creating a WebDAV Server Type Extension

Procedure

1. To add a new server type (also called Runtime in TIBCO Business Studio) you must extend com.tibco.xpd.deploy.core.serverTypes. By convention, servers are defined in separate eclipse plug-ins named com.tibco.xpd.deploy.server.servername. Therefore, create a new plug-in for the deployment server called com.tibco.xpd.deploy.server.webdav.
2. Next, provide the necessary dependencies:



3. Add the serverType extension that looks similar to the following:

```
<serverType  connectionFactory=
"com.tibco.xpd.deploy.server.webdav.WebDavConnectionFactory"
      id="com.tibco.xpd.deploy.server.webdav"
      name="Web DAV">
   <supportedRepository
         repositoryId="com.tibco.xpd.deploy.ui.WorkspaceRpository">
   </supportedRepository>
   <configParameter
         key="siteUrl"
         label="&Site URL:"
         name="Site URL"
         parameterType="string"
         required="true">
   </configParameter>
   <configParameter
         key="username"
         label="&User Name:"
         name="User Name"
         parameterType="string"
         required="false">
   </configParameter>
   <configParameter
         key="password"
         label="&Password:"
         name="Password"
         parameterType="password"
         required="false">
   </configParameter>
   <configParameter
         defaultValue="false"
         key="showSubfolders"
         label="Sho&w Sub Folders"
         name="Show Sub Folders"
         parameterType="boolean"
         required="true">
   </configParameter>
</serverType>
```

- The connectionFactory attribute points to an instance of ConnectionFactory interface, and its purpose is to provide the server connection implementation class for a particular server:

```
public class WebDavConnectionFactory implements ConnectionFactory {
    public Connection createConnection(Server server) {
        return new WebDavConnection(server);
    }
}
```

  Connection implementation is the central class of the server and is described in Creating the Server Type Connection Implementation.

- Server type must support at least one repository type. There are two repository types already defined in the com.tibco.deploy.ui plug-in, but you can also define your own repository type by extending the com.tibco.xpd.deploy.core.repositoryTypes extension. This tutorial uses the default, com.tibco.xpd.deploy.ui.WorkspaceRpository, which simply provides deployment modules directly from Eclipse workspace.
- Finally, all necessary server connection parameters are defined:

  siteUrl – URL to the WebDAV directory (also called collection) where files will be deployed.

  username, password – Authentication information for the WebDAV server.

  showSubfolders – Sets whether subfolders of the site should be shown. By default this parameter is false. If the repository is large, setting this parameter to true can cause refresh problems.

  Creating the Server Type Connection Implementation

Result

The server type Connection implementation class is responsible for communication with a remote server.

This includes the following:

- connecting to / disconnecting from server
- deploying modules
- providing and refreshing server elements
- updating server state
- performing server element operations

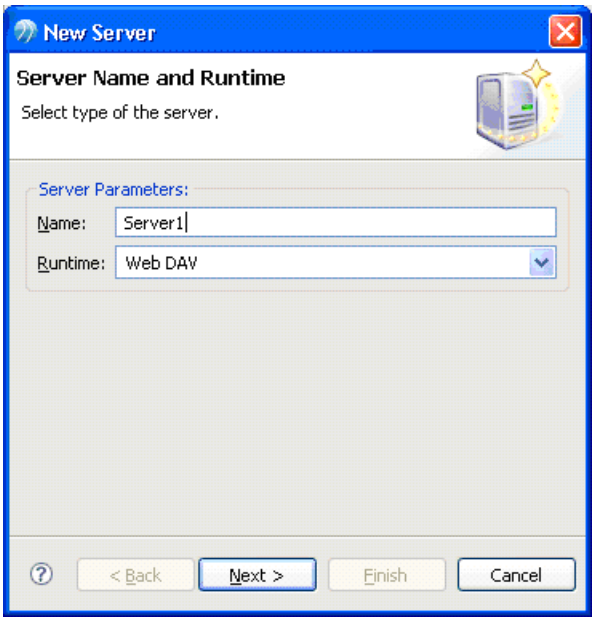
The simplest form of Connection implementation could look like this:

```
public class WebDavConnection implements Connection {
    private final Server server;
    public WebDavConnection(Server server) {
        this.server = server;
    }
    public void connect() throws ConnectionException {
        server.setServerState(ServerState.CONNECTED_LITERAL);
    }
    public void disconnect() throws ConnectionException {
        server.setServerState(ServerState.DISCONNECTED_LITERAL);
    }
    public boolean isConnected() throws ConnectionException {
        return server.getServerState() == ServerState.CONNECTED_LITERAL;
    }
public DeploymentStatus deployModule(String url) throws DeploymentException {
    return new DeploymentSimpleStatus(
          DeploymentSimpleStatus.Severity.OK, "", null);
    }
    public void refreshServerContent() throws ConnectionException {
    }
    public Server getServer() {
        return server;
    }
    public Object performServerElementOperation(ServerElement serverElement,
           Operation operation) throws DeploymentException {
```

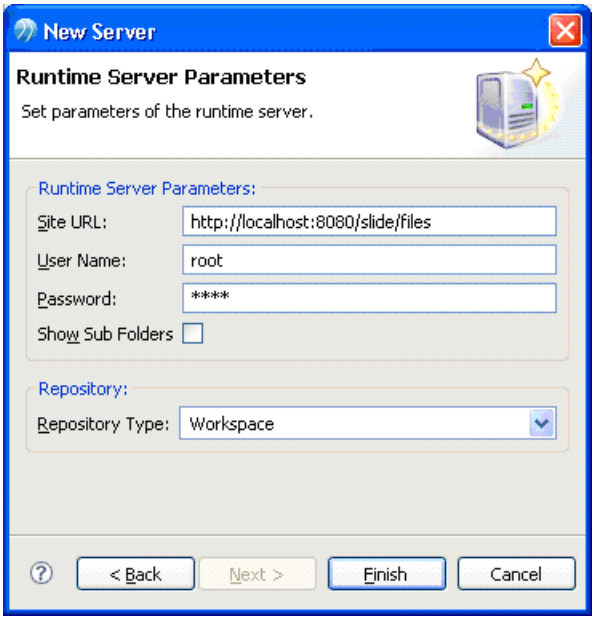
```
        return null;
    }
    public Object getAdapter(Class adapter) {
        return null;
    }
}
```

Although this implementation only changes the server state on a connect or disconnect action, it is sufficient for the purposes of the tutorial.

In TIBCO Business Studio, you should now be able to start the New Server Wizard and choose Web DAV as the Runtime. For example:



If you select Web DAV as the runtime, on the screen that follows, you can view the rendered parameters of the server and the assigned repository. The widgets to capture parameter values are automatically generated using the parameter descriptions provided in the extension.



After clicking the Finish button, you can view the new server created under the Deployment Servers in the Project Explorer. You can also invoke connect and disconnect actions from the context menu and as a result see the server state change.

Connecting to WebDAV server

To make a real connection to the server, you must implement the connect() method. The following example implementation of the method checks if the authentication information is correct for the provided site, and also checks that the site is a valid WebDAV resource.

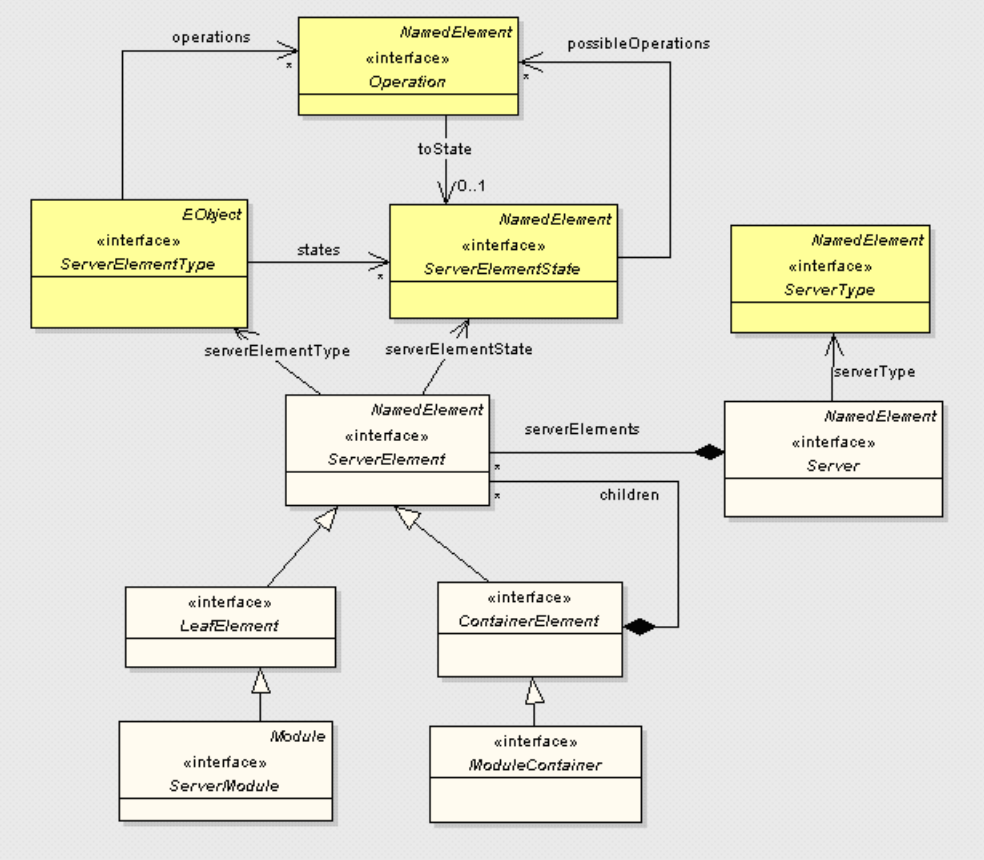
```
private CollectionHandle siteHandle;
public void connect() throws ConnectionException {
    ServerConfig config = server.getServerConfig();
    String siteUrl = config.getConfigParameter("siteUrl").
                        getValue().toString();
    String username = config.getConfigParameter("username").
                        getValue().toString();
    String password = config.getConfigParameter("password").
                        getValue().toString();
    try {
        password = EncryptionUtil.decrypt(password);
    } catch (IllegalArgumentException e) {
        throw new ConnectionException(
              "Connection failed. "
            + "Password has changed in the servers configuration file. "
            + "Please re-enter the server password.", e);
    }
    WebDAVFactory davFactory = new WebDAVFactory();
    HttpClient httpClient = new HttpClient();
    httpClient.setAuthenticator(new SimpleBasicAuthenticator(username,
            password));
    DAVClient davClient = new RemoteDAVClient(davFactory, httpClient);
    ILocator siteLocator = davFactory.newLocator(siteUrl);
    siteHandle = new CollectionHandle(davClient, siteLocator);
    try {
        if (siteHandle.canTalkDAV()) {
            server.setServerState(ServerState.CONNECTED_LITERAL);
        } else {
            throw new ConnectionException(
                     "Cannot connect to WebDAV site.");
        }
    } catch (DAVException e) {
        throw new ConnectionException(e);
    }
}
```

In this example, the server parameters' values are read from the server configuration. Then, we create davClient and siteLocator (which is responsible for locating DAV resource). Next we create CollectionHandle (the proxy for a remote resource) for the site and invoke the canTalkDAV() method. This method checks if the connection to the server can be made, and if the corresponding remote resource is a valid WebDAV collection resource.

The WebDAV is a request response protocol, so there is no explicit session associated with a connection. However, the HttpClient used by siteHandle does contain some session information (for example authentication information) that is sent with every request.

Providing Server Elements

The server can contain multiple ServerElement objects which represent objects on the server (usually existing modules). They are displayed in the Project Explorer under the server and can be arranged in a hierarchical structure:



The refreshServerContent() method from the Connection interface is responsible for providing and updating server elements. The following example shows an implementation of the method for a WebDAV server.

```
public void refreshServerContent() throws ConnectionException {
    if (isConnected() && siteHandle != null) {
        try {
            refreshServer(siteHandle);
        } catch (DAVException e) {
            server.getServerElements().clear();
            disconnect();
            throw new ConnectionException(e);
        }
    } else {
        server.getServerElements().clear();
    }
}
private void refreshServer(CollectionHandle site) throws DAVException {
    boolean showSubfolders = Boolean.parseBoolean(WebDavConstants
            .getConfigParamValue(server.getServerConfig(),
                    WebDavConstants.SHOW_SUBFOLDERS));
    HashSet<String> existingFileNames = new HashSet<String>();
    HashSet<String> existingDirNames = new HashSet<String>();
    Set<AbstractResourceHandle> members = site.getMembers();
    for (Iterator iter = server.getServerElements().iterator(); iter
            .hasNext();) {
        ServerElement se = (ServerElement) iter.next();
        String name = se.getName();
        if (se instanceof ModuleContainer) {
```

```
            boolean exist = false;
            for (AbstractResourceHandle member : members) {
                if (isCollectionMember(member)
                        && name.equals(getMemberName(member))) {
                    if (showSubfolders) {
                        refreshCollection((CollectionHandle) member,
                                (ContainerElement) se);
                    }
                    exist = true;
                    existingDirNames.add(name);
                }
            }
            if (!exist) {
                iter.remove();
            }
        } else if (se instanceof ServerModule) {
            boolean exist = false;
            for (AbstractResourceHandle member : members) {
                if (!isCollectionMember(member)
                        && name.equals(getMemberName(member))) {
                    setModuleProperties((ServerModule) se, member);
                    exist = true;
                    existingFileNames.add(name);
                }
            }
            if (!exist) {
                iter.remove();
            }
        }
    }
    for (AbstractResourceHandle member : members) {
        String memberName = getMemberName(member);
        if (isCollectionMember(member)) {
            if (!existingDirNames.contains(memberName)) {
                ModuleContainer moduleContainer = DeployFactory.eINSTANCE
                        .createModuleContainer();
                moduleContainer.setName(memberName);
                if (showSubfolders) {
                    refreshCollection((CollectionHandle) member,
                            moduleContainer);
                }
                server.getServerElements().add(moduleContainer);
            }
        } else {
            if (!existingFileNames.contains(memberName)) {
                ServerModule module = DeployFactory.eINSTANCE
                        .createServerModule();
                module.setName(memberName);
                setModuleProperties(module, member);
                server.getServerElements().add(module);
            }
        }
    }
}
```

During the method invocation, we connect to the WebDAV server and obtain remote resources' information. Then we create corresponding server modules and/or update their state and properties.

When the user disconnects from the server, all server elements should be removed.

In a future implementation, the server elements could be enhanced to better support hierarchical structures of server objects.

Deploying Modules

To implement modules deployment we will have to do the following:

- Implement DeploymentStatus deployModule(String url) method for the Connection implementation.
- Provide a deployment wizard.

Implementing the deployModule Method

The example in this section shows an implementation of the deployModule method for a WebDAV server. This method is responsible for the deployment of a single file to the WebDAV site.

```
public DeploymentStatus deployModule(String url)
                 throws DeploymentException {
    Assert.isNotNull(siteHandle);
    Assert.isTrue(url != null && url.trim().length() > 0);
    InputStream inputStream = null;
    String deploymentMsg = "Deploying: " + url;
    try {
        DAVClient client = siteHandle.getDAVClient();
        String remoteUrl = getSiteRelativeURL(siteHandle, url);
        ILocator resourceLocator = client.getDAVFactory().newLocator(
                remoteUrl);
        URL localUrl = new URL(url);
        inputStream = localUrl.openStream();
        IResponse response = client.put(resourceLocator, client
                .getDAVFactory().newContext(), inputStream);
        int statusCode = response.getStatusCode();
        String statusMessage = response.getStatusMessage();
        String responseMessage = "\n" + statusCode + ':' + statusMessage;
        if (statusCode == IResponse.SC_CREATED
                || statusCode == IResponse.SC_NO_CONTENT) {
            return new DeploymentSimpleStatus(
                    DeploymentSimpleStatus.Severity.OK, deploymentMsg
                            + responseMessage, null);
        } else {
            return new DeploymentSimpleStatus(
                    DeploymentSimpleStatus.Severity.ERROR, deploymentMsg
                            + responseMessage, null);
        }
    } catch (IOException e) {
        return new DeploymentSimpleStatus(
                DeploymentSimpleStatus.Severity.ERROR, deploymentMsg, e);
    } finally {
        try {
            inputStream.close();
        } catch (IOException e) {
            // TODO Log error
            e.printStackTrace();
        }
    }
}
```

The method's parameter is a string that contains the URL of the file in the deployment repository (also called the inquiry URL). Because we use "Workspace Repository" this URL is the same as the URL of the local file. Before the module is deployed, it is published to the server repository and the repository is asked to provide the inquiry URL for the published module. This inquiry URL is passed as a parameter to the deployModule method, and it is used by the server to access the module from the repository (it could be different to the local module URL).

Providing the Deployment Wizard

This section describes how to provide a deployment wizard and associate it with the WebDAV server type by implementing the com.tibco.xpd.deploy.ui.deployWizards extension point.

```
<extension
  point="com.tibco.xpd.deploy.ui.deployWizards">
  <deployWizard
   class="com.tibco.xpd.deploy.server.webdav.ui.WorkspaceFileDeployWizard"
   id="com.tibco.xpd.deploy.server.webdav.workspaceFileWizard"
   name="Workspace File"
   serverTypeId="com.tibco.xpd.deploy.server.webdav"/>
</extension>
```

The main purpose of a deployment wizard is to provide a list of URLs for local modules. To do this, the wizard class must implement IDeployWizard. The most important method in this interface is List<URL> getModulesUrls(), which (after the wizard finishes) should return a list of modules' local URLs.

Implementing Operations for ServerElements

To implement operations for server elements, you must define the following:

- Types of elements
- Possible states for a defined element type
- Operations that can be performed
- Operations that can be performed when an element is in a particular state

It is also necessary to set the correct type and state for every server element created when calling the refreshServer method.

The following code fragment shows how to create and configure all necessary elements to implement a WebDAV module (file) delete operation.

```
private OperationImpl deleteFileOperation;
private ServerElementType fileType;
private ServerElementState publishedState;
private void initialiseServerElementTypes() {
    DeployFactory f = DeployFactory.eINSTANCE;
    fileType = f.createServerElementType();
    // states. States should not be shared between different element types
    publishedState = f.createServerElementState();
    publishedState.setName("Published");
    // all states server element type could be in
    fileType.getStates().add(publishedState);
    // operations
    deleteFileOperation = new OperationImpl() {
        @Override
        public Object execute(ServerElement serverElement)
                throws DeploymentException {
            try {
                return deleteRemoteFile(serverElement);
            } finally {
                refreshServerContent();
            }
        }
    };
    deleteFileOperation.setName("Delete");
    // all available operations for server element type
    fileType.getOperations().add(deleteFileOperation);
    // possible operations for states
    publishedState.getPossibleOperations().add(deleteFileOperation);
}
```

Also when you create modules using the refreshServer method, you must set an element's type and state as follows:

```
serverElement.setServerElementType(fileType);
serverElement.setServerElementState(publishedState);
```

Summary

This tutorial showed how to use the TIBCO Business Studio Deployment Framework to create a plug-in for the deployment of a workspace file to a WebDAV server. It also described how to retrieve server objects, information about them, and how to implement operations performed on these objects.

The following features are available, but were not demonstrated by this tutorial:

- The ability of the deployment framework to allow you to create additional module repository types.
- The reuse of configuration information in a client runtime. This is useful, for example, if many server types use common configuration data or a common platform on the client side. This common

configuration information can be stored by the client runtime which can be shared by multiple servers. For more information about runtimes, see Defining a Runtime (Optional).

- "On save" module deployment, in which a module is automatically deployed when the content of the module changes.

TIBCO Documentation and Support Services

How to Access TIBCO Documentation

Documentation for TIBCO products is available on the TIBCO Product Documentation website, mainly in HTML and PDF formats.

The website is updated frequently and is more current than any other documentation included with the product.

Product-Specific Documentation

The following documentation for TIBCO Business Studio is available on the TIBCO Business Studio Product Documentation page:

- TIBCO Business Studio™ Release Notes
- TIBCO Business Studio™ Concepts
- TIBCO Business Studio™ Modeling User's Guide
- TIBCO Business Studio™ - Analyst Edition User's Guide
- TIBCO Business Studio™ - BPM Implementation
- TIBCO Business Studio™ Forms User's Guide
- TIBCO Business Studio™ Simulation User's Guide
- TIBCO Business Studio™ Customization
- TIBCO Business Studio™ - Analyst Edition Installation
- TIBCO Business Studio™ - BPM Edition Installation
- TIBCO Business Studio™ iProcess to BPM Conversion

How to Contact TIBCO Support

Get an overview of TIBCO Support. You can contact TIBCO Support in the following ways:

- For accessing the Support Knowledge Base and getting personalized content about products you are interested in, visit the TIBCO Support website.
- For creating a Support case, you must have a valid maintenance or support contract with TIBCO. You also need a user name and password to log in to TIBCO Support website. If you do not have a user name, you can request one by clicking **Register** on the website.

How to Join TIBCO Community

TIBCO Community is the official channel for TIBCO customers, partners, and employee subject matter experts to share and access their collective experience. TIBCO Community offers access to Q&A forums, product wikis, and best practices. It also offers access to extensions, adapters, solution accelerators, and tools that extend and enable customers to gain full value from TIBCO products. In addition, users can submit and vote on feature requests from within the TIBCO Ideas Portal. For a free registration, go to TIBCO Community.

Legal and Third-Party Notices